I.T. CARLOW, SCHOOL OF SCIENCE AND ENGINEERING

Software Design

Tradesmen.ie

Ilya Biryukov. C00117434@itcarlow.ie, ilya@curiousdeveloper.net

11/01/2011

Mentor: Joseph Kehoe

This document describes the software design for tradesmen.ie application by providing detailed uses cases, a domain model, system sequence diagrams, a class diagram and a database architecture

Contents

System Overview

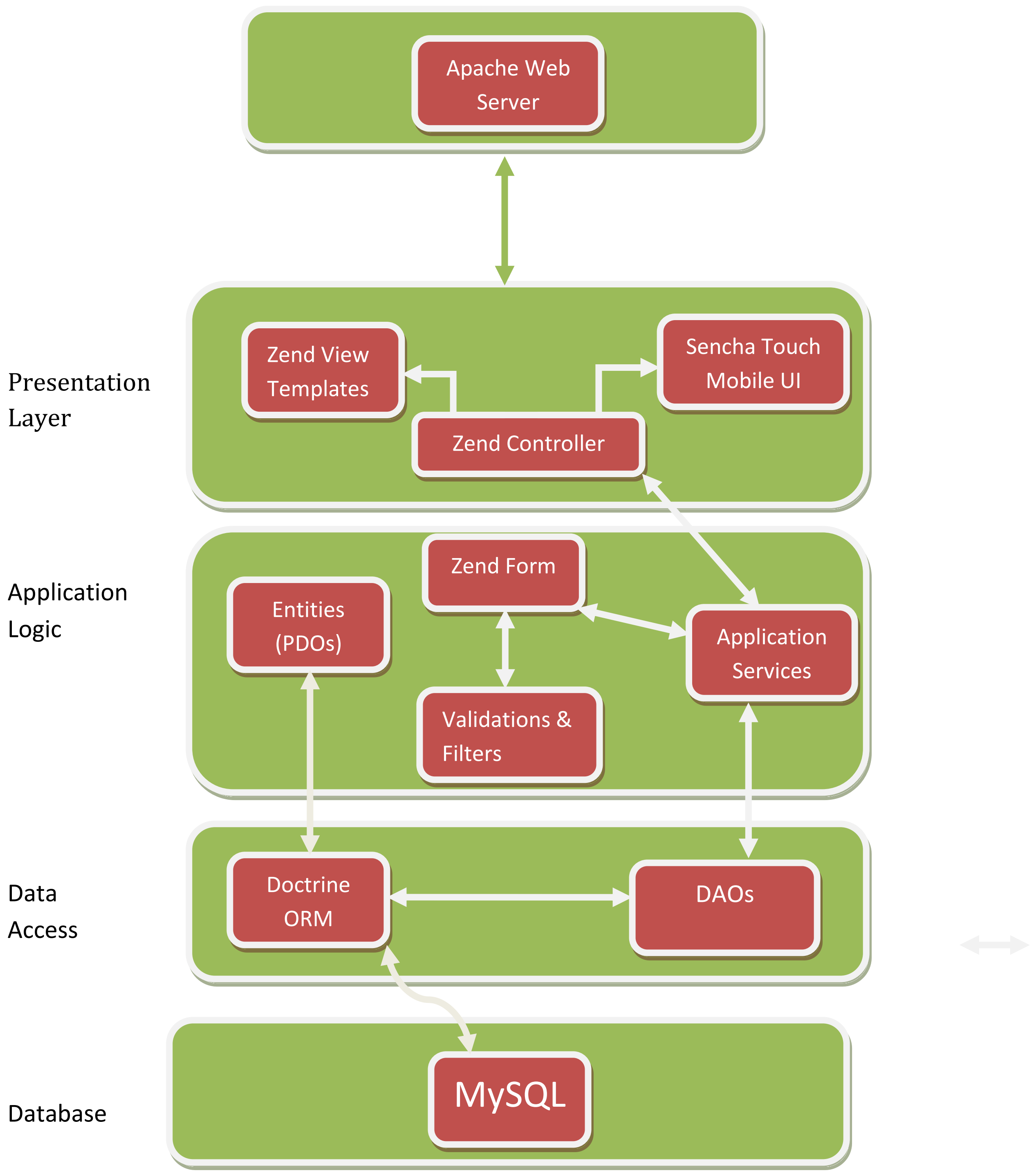


Figure 1. System architecture overview

The system essentially consists of three-tier (layer) architecture with one additional external layer on top (apache) and one additional external system below (MySQL) for persistent storage. The system fully implements the Model View Controller Pattern with the Controller and View situated and at the Presentation Layer, and the Model spread across the Application Logic and Data Access layers.

At presentation layer, Zend Controller acts as glue to the whole application. It receives and replies to HTTP page requests. It takes a decision as to which template engine to turn on – Sencha Touch for Android and iOS users and Zend View for everybody else including desktop browsers and unsupported mobile browsers. Furthermore, it consumes the services API which contains all business logic and manages access to data.

At application logic layer, Entities (PDOs – PHP Data Objects) contain all information that is associated with any given Domain Object inside the system. They also contain some business logic and guarantee that only allowed data transformations happen by governing and maintain the list of allowed actions. For instance, a User Entity will ensure that User's password conforms to standards defined in the system or will propagate an error message back to the Service through its call stack. Zend Input Validators are used to ensure the validity of data both at Entities Level and at Zend Form level where Zend Form is a collection of classes that defined HTML forms and all of their attributes and elements which are used to gather data from the user and validate it.

At data access layer, there are components that ensure transparent persistency of Entities and also data retrieval and its transparent mapping to entities. Doctrine ORM (Object Relational Mapper) is an open source library that sits on top of a powerful database abstraction layer. Doctrine ORM implements DQL (Doctrine Query Language) that is used to run queries on Entities, pass the query to DAL which sends a transformed native SQL query to the database in a format that a particular database understands. Doctrine DBAL provides a complete abstraction from specifics of a database and its implementation, allowing us to swap the database at any time without rewriting a single line of code.

DAO (Data Access Object) is a famous and widely used pattern that acts as a dumb-proxy for the ORM (Doctrine ORM in our case). It removes DQL queries from Zend Controller gathering them in one place and allowing Controllers to focus on the presentation layer. DAOs provide a set of common functionality that a Controller can consume in order to retrieve or persist entities. On the other hand, DAOs talk to the ORM and request information from in form of Entities and persist information to it, again, in form of Entities.

And at the very bottom layer, we have the physical relational fully ACID compliant database - MySQL.

Use Case Diagrams

The system contains three major components:

1. Jobs component – describes who jobs can be created, edited, deleted and browsed
2. Accounts component – deals with user accounts and their management by system administrators
3. Bids component – describes how bids can be placed, amended and deleted for any job.

The summary diagram is provided below for each of the components which the detailed description of each use case is given in the next section.

Jobs Use Case Diagram

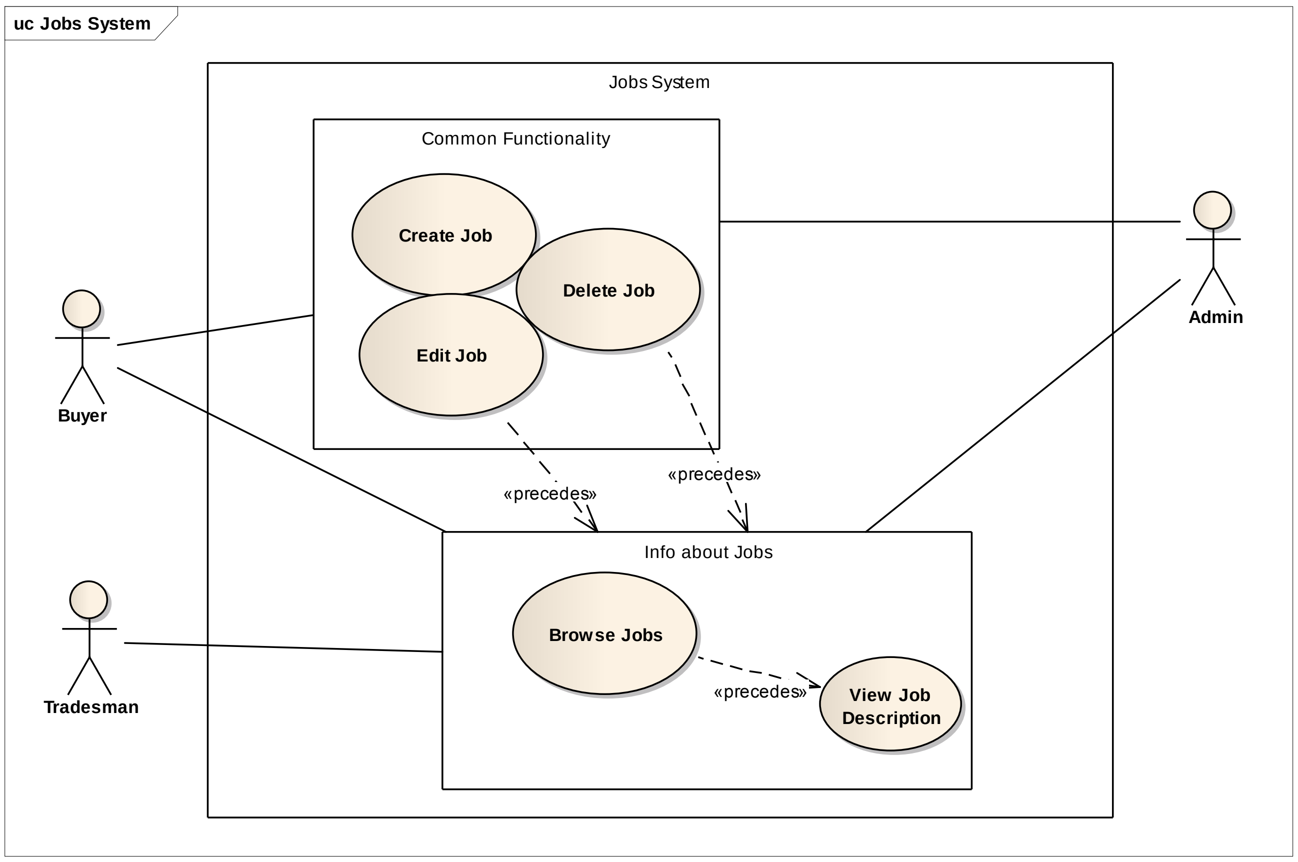


Figure 2. Jobs Uses Cases Diagram

Accounts Use Case Diagram

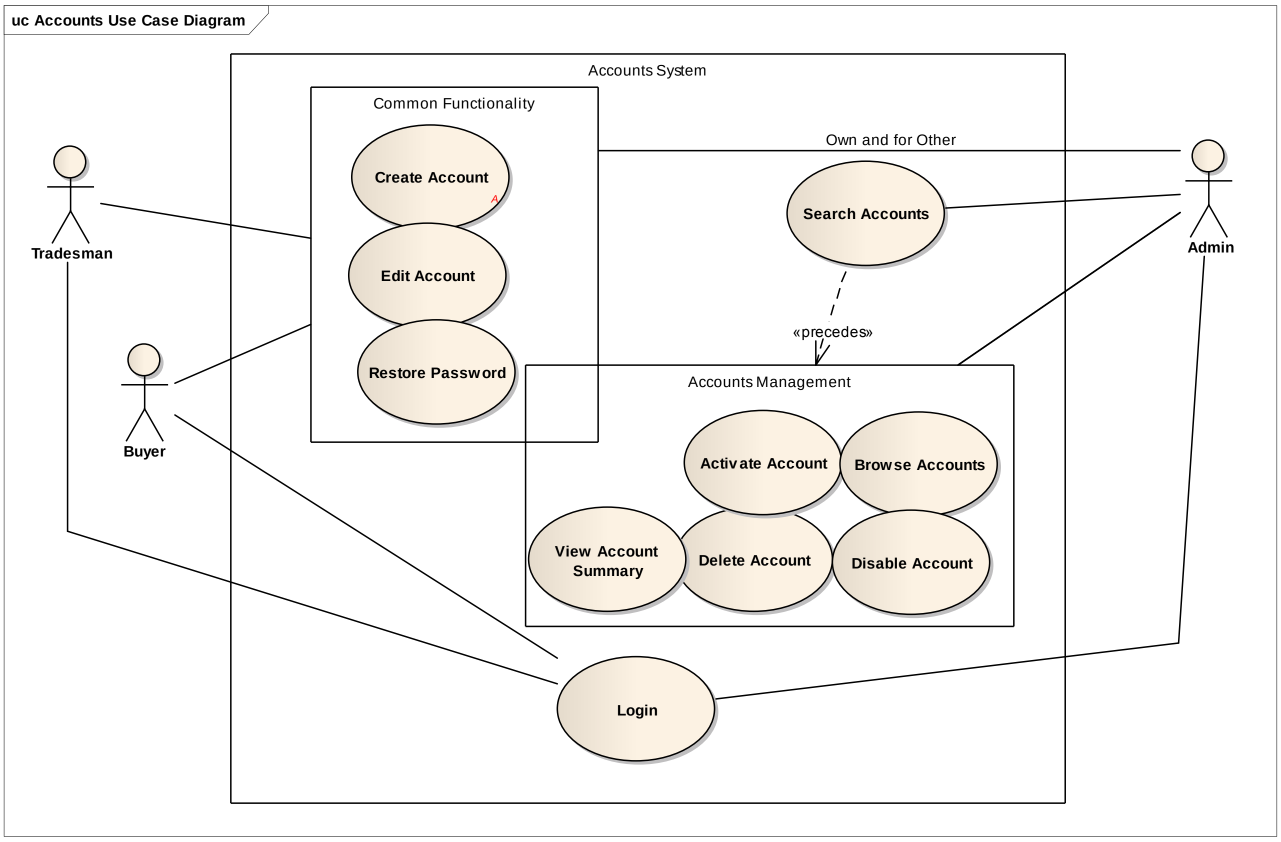


Figure 3. Accounts Uses Cases Diagram

Bids Use Case Diagram

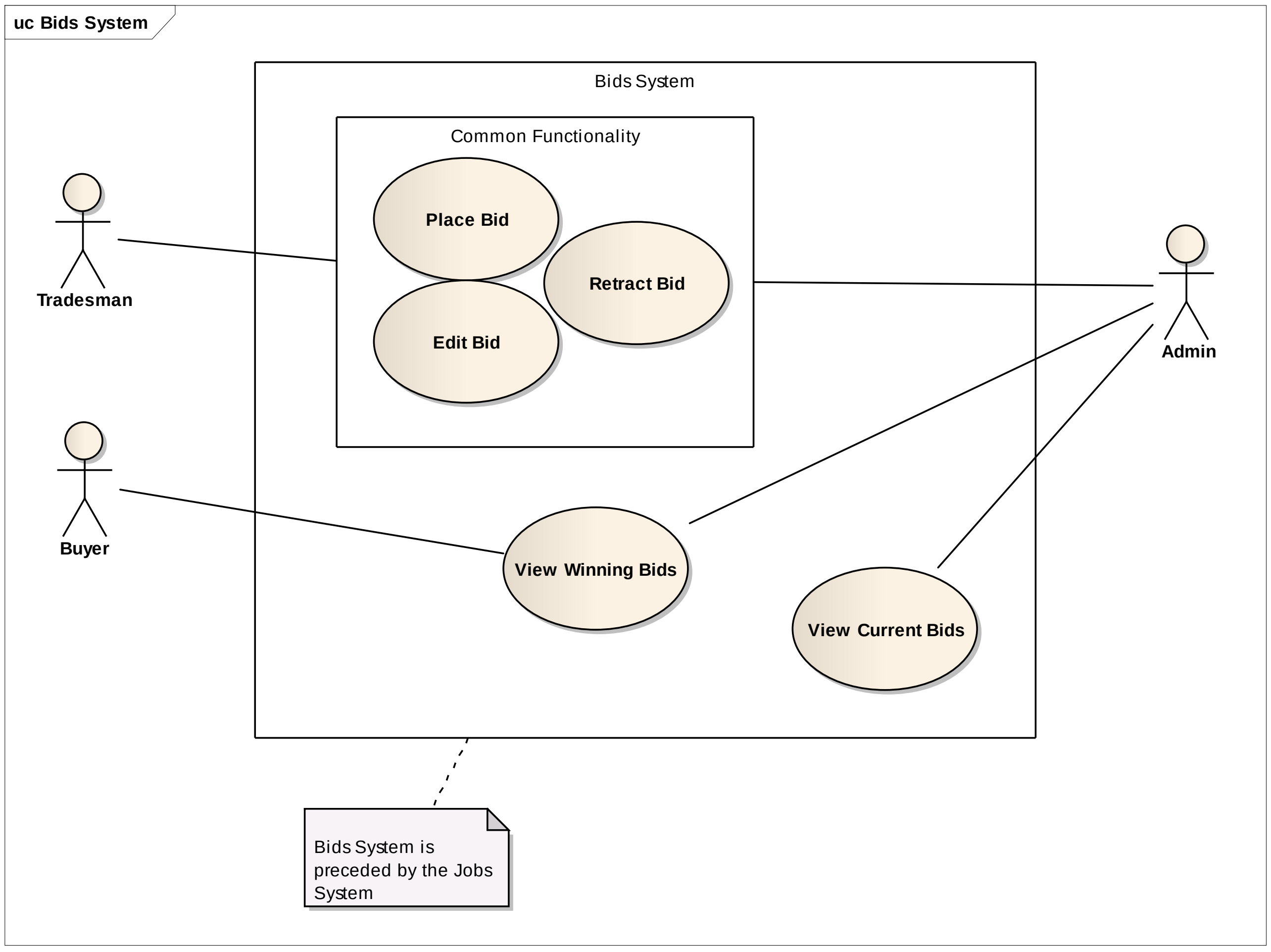


Figure 4. Bids Uses Cases Diagram

Detailed Use Cases

Accounts Use Cases

Create Account

Actor: Buyer

Lever: User Goal

Preconditions:

- The actor is not logged in.

Success Guarantee:

- A new *BuyerAccount* created with a unique user ID.
- The system displays a confirmation to the user and sends out a confirmation email.

Main Success Scenario:

1. The actor chooses to register as a buyer
2. The system checks that the actor is not logged in and displays *Buyer Registration Form*
3. The actor fills in the *First Name*
4. The actor fills in the *Last Name*
5. The actor fills in the *Phone Number*
6. The actor fills in the *Town*
7. The actor selects the *Country*
8. The actor fills in the *Email*
9. The actor confirms the *Email*
10. The actor accepts *The Terms and Conditions*
11. The actor clicks the *Register Button*
12. The system checks the provided information, ensures that email address is unique, creates a new Buyer Account and sends out an email confirmation with instructions to activate the account and redirects the actor to confirmation page.

Alternative Flows:

2a. The actor is already logged in

1. The system displays an error message (`ALREADY_LOGGED_IN`) notifying that the actor is already logged in.
2. The system displays a logout link.
3. The actor clicks the logout link
4. The system terminates actors session
5. The system notifies the actor of session termination
6. The actor resumes registration process from step 10.

12a. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 10

12b. *Email* is already registered.

1. The system displays an error notifying the actor that such email is already registered
2. The system also show links to Login Page and Password Restore Page.
3. The actor corrects the error and resumes from step 10
   3a. The actor navigates away to other page, possibly to login or restore password.

Actor: Tradesman

Lever: User Goal

Preconditions:

- The actor is not logged in.

Success Guarantee:

- A new *TradesmanAccount* created with a unique user ID.
- The system displays a confirmation to the actor and sends out a confirmation email.

Step 1. Personal Information

Main Success Scenario:

1. The actor chooses to register as a tradesman
2. The system displays *Tradesman Registration Form Step 1*
3. The actor fills in the *First Name*
4. The actor fills in the *Last Name*
5. The actor fills in the *Phone Number*
6. The actor fills in the *Town*
7. The actor selects in the *County*
8. The actor fills in the *Email*
9. The actor confirms the *Email*
10. The actor clicks the *Continue* button
11. The system checks the provided information and redirects the actor to *Step 2 Skills* page

Alternative Flows:

2a. The actor is already logged in

1. The system displays an error message (`ALREADY_LOGGED_IN`) notifying that the actor is already logged in.
2. The system displays a logout link.
3. The actor clicks the logout link
4. The system terminates actors session
5. The system notifies the actor of session termination
6. The actor resumes registration process from step 10.

11a. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 10

11b. *Email* is already registered.

1. The system displays an error notifying the actor that such email is already registered
2. The system also show links to Login Page and Password Restore Page.
3. The actor corrects the error and resumes from step 10
   3a. The actor navigates away to other page, possibly to login or restore password.

Step 2. Skills

Main Success Scenario:

1. The actor arrives to this page from previous page or next page
2. The system displays *Tradesman Registration Form Step 2.*
3. The actor selects the *Qualifications* that the actor holds
4. The actor fills in *Additional Qualifications*
5. The actor selects *Counties* from which the actor wishes to accept jobs
6. The actor fills in the *Brief Description of the Work* they do.
7. The actor clicks the *Continue* button.
8. The system checks the provided information and redirects the actor to *Step 3 Rates* page

Alternative Flows:

2a. Session Timed out / Form Data Missing

1. The system displays an error message (`SESSION_TIMEDOUT_REGISTRATION`) notifying that actor's session has timed out and they need to restart the registration process.
2. The actor goes back to *Step 1 Personal Information*

2b. The actor is already logged in

1. The system redirects the actor the *Step 1 Personal Information* page

3a. The actor wishes to amend details on the previous page

1. The actor clicks the *Go Back* button.
2. The system redirects the actor to *Step 1 Personal Information* page.

8a. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 7

Step 3. Rates

Main Success Scenario:

1. The actor arrives to this page from previous page
2. The system displays *Tradesman Registration Form Step 3*
3. The actor fills in the *Hourly Rate*
4. The actor fills in the *Daily Rate*
5. The actor fills in the *Fixed Rates*
6. The actor fills in the *Minimum Job Value*
7. The actor fills in the *Maximum Job Value*
8. The actor checks the checkbox accepting the *Terms & Conditions*
9. The actor clicks the *Finish Registration* button
10. The system checks entered information and redirects the actor to *Step 4 Confirmation* page.

Alternative Flows:

2a. Session Timed out / Form Data Missing

1. The system displays an error message (`SESSION_TIMEDOUT_REGISTRATION`) notifying that actor's session has timed out and they need to restart the registration process.
2. The actor goes back to *Step 1 Personal Information*

2b. The actor is already logged in

2. The system redirects the actor the *Step 1 Personal Information* page

3a. The actor wishes to amend details on the previous page

1. The actor clicks the *Go Back* button.
2. The system redirects the actor to *Step 2 Skills* page.

8a. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 9

Step 4. Registration Confirmation

Main Success Scenario:

1. The actor arrives to this page from previous page
2. The system creates new *Tradesman Account* and emails the tradesman a verification letter with steps to activate the account.
3. The system displays a confirmation message confirmation the successful registration and explains the steps necessary to activate the account.

Alternative Flows:

2a. Session Timed out / Form Data Missing

1. The system displays an error message (`SESSION_TIMEDOUT_REGISTRATION`) notifying that actor's session has timed out and they need to restart the registration process.
2. The actor goes back to *Step 1 Personal Information*

2b. The actor is already logged in

1. The system redirects the actor the *Step 1 Personal Information* page

Login

Actor: Buyer, Tradesman, Admin

Lever: User Goal

Preconditions:

- The actor is not logged in.
- The actor is already registered.
- Actor's account is activated
- Actor's account is not disabled

Success Guarantee:

- A new user session is created
- User's privileges are loaded for existing resources and stored in the session

Main Success Scenario:

1. The actor fills in the *Email*
2. The actor fills in the *Password*
3. The actor clicks on the *Login* button
4. The system redirects the actor to login page, checks entered information, finds the user record, creates a new user session and loads actor's privileges and then redirects back to the original page.

Alternative Flows:

4a. The actor is already logged in

1. The system displays an error message (`ALREADY_LOGGED_IN`) notifying that the actor is already logged in.

4b. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 3

4c. The Email/Password Combination is not found

1. The system displays an error message (`LOGIN_FAILED`).
2. Use case restarts from step 1

4d. The account is not activated.

1. The system displays an error message (`ACCOUNT_NOT_ACTIVATED`) to notify the actor that the account needs to activated before log in function becomes available
2. The system displays a link to resend activation email.

a. The actor clicks the resend activation email link
b. The system sends a new activation email and displays a confirmation

4e. The account disabled

1. The system displays an error message (`ACCOUNT_DISABLED`) to notify the actor that the account is disabled.

Logout

Actor: Buyer, Tradesman, Admin

Lever: User Goal

Preconditions:

- The actor is logged in.

Success Guarantee:

- Actor’s Session is destroyed, and the actor is redirected to the main page

Main Success Scenario:

1. The actor clicks the logout link at the top of the page
2. The system checks that the actor is logged in, destroys actor’s session and redirects the actor to the main page

Alternative Flows:

None

Search Accounts

Actor: Admin

Level: Admin Goal

Preconditions:

- The actor is Admin

Success Guarantee:

- A search result is displayed with the list of users found that matched the criteria or an error message is displayed notifying the actor that the search did not match any users.

Main Success Scenario:

1. The actor navigates to *User Management* page
2. The system displays *Search Accounts* Form
3. The actor fills in any number of fields and clicks the search button
4. The system retrieves the list of accounts that matched the entered criteria and displays the list of results, hiding the search form

Alternative Flows:

4a. No data was provided

1. The system displays the error message (USER_SEARCH_NO_DATA) notifying the actor that they need to fill in at least one field for the search to be performed
2. The actor corrects the errors
3. The actor resumes from step 3

4b. No matches found

1. The system displays an error message (USER_SEARCH_NO_MATCH) notifying the actor that there were no matching users found
2. The actor amends the search criteria and performs the search again, restarting from step 3
    a. The actor navigates away from the page.

Restore Password

Actor: Buyer, Tradesman, Admin

Lever: User Goal, Admin Goal

Preconditions:

- The actor is already registered.

Success Guarantee:

- A new password is created for the actor
- An confirmation message is displayed
- An email is sent to the actor with the new password

Main Success Scenario:

1. The actor navigates to *Restore Password* page
2. The system displays the *Restore Password* form
3. The actor fills in the *Email* address
4. The actor fills in the *Phone Number*
5. The actor clicks the *Get New Password* button
6. The system checks entered information, generates new password, updates the *User Account* with the password, sends out an email with the new password and displays a confirmation.

Alternative Flows:

6a. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 5

4a. The account is not found

1. The system displays an error message (`ACCOUNT_NOT_FOUND`) notifying the actor that his account was not found.

Edit Account

Actor: Admin

Level: Admin Goal

Preconditions:

- The actor is administrator
- The *User Account* exists

Success Guarantee:

- The *User Account* record is updated
- An confirmation message is displayed

Main Success Scenario:

1. The actor is browsing the list of accounts and clicks an edit button that corresponds to a given *User Account*
2. The system checks that the actor has permissions to edit the *User Account,* fetches the user account and displays an *Edit User Account* form
3. The actor amends any number of fields and clicks the *Edit* button
4. The system checks that the actor has permissions to edit the *User Account,* ensures that required data is present, updates the *User Account* and displays a confirmation message, redirecting the user back to the List of Search Results

Alternative Flows:

2a/4a. The actor doesn't have privileges to edit accounts

1. The system displays an error message (NO_PRIV_EDIT_ACCOUNT) notifying the actor he doesn't have sufficient permission to perform the action.

2b/4b. The account is not found

1. The system displays an error message (ACCOUNT_NOT_FOUND) notifying the actor that his account was not found.

4c. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 3

Delete Account

Actor: Admin

Lever: Admin Goal

Preconditions:

- The actor is administrator
- The *User Account* exists

Success Guarantee:

- The *User Account* record is deleted
- An confirmation message is displayed

Main Success Scenario:

1. The user clicks *Delete Account* button from the *Browse Accounts* or *Full Profile* page
2. The system displays the *Delete Account Confirmation* dialog
3. The actor confirms the action by clicking the *Yes* button
4. The system deletes the an appropriate *User Account* record and displays a confirmation

Alternative Flows:

3a. The actor rejects the action

1. The actor clicks the *Cancel* button
2. The system hides the *Delete Account Confirmation* dialog and takes no further action

4a. The actor doesn't have privileges to delete accounts

1. The system displays an error message (NO_PRIV_DELETE_ACCOUNT) notifying the actor he doesn't have sufficient permission to perform the action.

4b. The account is not found

1. The system displays an error message (`ACCOUNT_NOT_FOUND`) notifying the actor that his account was not found.

Disable Account

Actor: Admin

Lever: Admin Goal

Preconditions:

- The actor is administrator
- The *User Account* exists
- The account is enabled

Success Guarantee:

- *Disabled* flag in The *User Account* is set to true
- An confirmation message is displayed

Main Success Scenario:

1. The user clicks *Disable Account* button from the *Browse Accounts* or *Full Profile* page
2. The system displays the *Disable Account Confirmation* dialog
3. The actor confirms the action by clicking the *Yes* button
4. The system disables the account by updating an appropriate *User Account* record and displays a confirmation

Alternative Flows:

3a. The actor rejects the action

1. The actor clicks the *Cancel* button
2. The system hides the *Disable Account Confirmation* dialog and takes no further action

4a. The actor doesn't have privileges to delete accounts

1. The system displays an error message (NO_PRIV_DISABLE_ACCOUNT) notifying the actor he doesn't have sufficient permission to perform the action.

4b. The account is not found

1. The system displays an error message (ACCOUNT_NOT_FOUND) notifying the actor that his account was not found.

4c. The account is already disabled

1. The system does not alter any data, but displays the confirmation

Enable Account

Actor: Admin

Lever: Admin Goal

Preconditions:

- The actor is administrator
- The *User Account* exists
- The account is disabled

Success Guarantee:

- *Disabled* flag in The *User Account* is set to false
- An confirmation message is displayed

Main Success Scenario:

1. The user clicks *Enable Account* button from the *Browse Accounts* or *Full Profile* page
2. The system displays the *Enable Account Confirmation* dialog
3. The actor confirms the action by clicking the *Yes* button
4. The system disables the account by updating an appropriate *User Account* record and displays a confirmation

Alternative Flows:

3a. The actor rejects the action

1. The actor clicks the *Cancel* button
2. The system hides the *Enable Account Confirmation* dialog and takes no further action

4a. The actor doesn't have privileges to delete accounts

1. The system displays an error message (NO_PRIV_ENABLE_ACCOUNT) notifying the actor he doesn't have sufficient permission to perform the action.

4b. The account is not found

1. The system displays an error message (ACCOUNT_NOT_FOUND) notifying the actor that his account was not found.

4c. The account is already Enabled

1. The system does not alter any data, but displays the confirmation

Resend Activation Email

Actor: Buyer, Tradesman, Admin

Lever: User Goal, Admin Goal

Preconditions:

- The actor is administrator
- The *User Account* exists
- The account is not activated

Success Guarantee:

- A new confirmation email with a *Activation Code* is generated and sent to user's email
- An confirmation message is displayed

Main Success Scenario:

1. The user clicks *Resend Activation Email* button from the *Browse Accounts* or *Full Profile* page
2. The system displays the *Resend Activation Email* dialog
3. The actor confirms the action by clicking the *Yes* button
4. The system generates new Activation Code for the appropriate *User Account* record, send an email with the activation link and displays a confirmation

Alternative Flows:

3a. The actor rejects the action

1. The actor clicks the *Cancel* button
2. The system hides the *Enable Account Confirmation* dialog and takes no further action

4a. The actor doesn't have privileges to resend validation emails accounts

1. The system displays an error message (NO_PRIV_REMAIL_ACCOUNT) notifying the actor he doesn't have sufficient permission to perform the action.

4b. The account is not found

1. The system displays an error message (ACCOUNT_NOT_FOUND) notifying the actor that his account was not found.

4c. The account is already Activated

1. The system does not alter any data and displays an error message (ACCOUNT_ALREADY_ACTIVATED) notifying the actor that the account is already activated.

Activate Account

Actor: Admin

Lever: Admin Goal

Preconditions:

- The actor is administrator
- The *User Account* exists
- The account is not activated

Success Guarantee:

- The Activated property from User Account record is set to true
- An confirmation message is displayed

Main Success Scenario:

1. The user clicks *Activate Account* button from the *Browse Accounts* or *Full Profile* page
2. The system displays the *Activate Account Confirmation* dialog
3. The actor confirms the action by clicking the *Yes* button
4. The system activates the account by updating an appropriate *User Account* record and displays a confirmation

Alternative Flows:

3a. The actor rejects the action

1. The actor clicks the *Cancel* button
2. The system hides the *Activate Account Confirmation* dialog and takes no further action

4a. The actor doesn't have privileges to activate accounts

1. The system displays an error message (NO_PRIV_ACTIVATE_ACCOUNT) notifying the actor he doesn't have sufficient permission to perform the action.

4b. The account is not found

1. The system displays an error message (ACCOUNT_NOT_FOUND) notifying the actor that his account was not found.

4c. The account is already Activated

1. The system does not alter any data, but displays the confirmation

View Full Profile

Actor: Admin

Lever: Admin Goal

Preconditions:

- The actor is administrator
- The *User Account* exists

Success Guarantee:

- The information about the chosen account is displayed or an error message is given

Main Success Scenario:

1. The user clicks *View Profile* button from the *Browse Accounts* or *Full Profile* page
2. The system checks that the actor has permissions to view user profiles, retrieves the *User Record* for the chosen account and displays the information about the account

Alternative Flows:

2a. The actor doesn't have privileges to view accounts

2. The system displays an error message (NO_PRIV_VIEW_ACCOUNT) notifying the actor he doesn't have sufficient permission to perform the action.

2b. The account is not found

2. The system displays an error message (ACCOUNT_NOT_FOUND) notifying the actor that his account was not found.

Jobs Use Cases

Browse Jobs List

Actor: Buyer, Tradesman, Admin

Lever: User Goal

Preconditions: None

Success Guarantee:

- A List of jobs is active jobs is displayed or an error message

Main Success Scenario:

1. The actor navigates to the *Jobs List* page.
2. The system fetches a list of currently active jobs
3. The system displays the jobs in a paginated manner.

Alternative Flows:

3a. No Active Jobs Available

1. The system displays an error message (NO_ACTIVE_JOBS) to notify the actor that there are currently no active jobs on the system.

View Full Job Description

Actor: Buyer, Tradesman, Admin

Lever: User Goal

Preconditions:

- The job that is being accessed exists

Success Guarantee:

- The system displays the details about the job or notifies the user that the requested job was not found

Main Success Scenario:

1. The user clicks on a particular job on the *Jobs List* page or otherwise navigates to *View Full Job Description* page.
2. The system retrieves data for the requested job
3. The system displays the information about the job

Alternative Flows:

2a. The actor is admin

1. The system also retrieves the list of current bids

3a. The actor is admin

1. The system also displays the list of current bids

3b. No Such Job Found

1. The system displays an error message (REQ_JOB_NOT_FOUND) to notify the actor that the requested job was not found.

3c. The Actor is not logged in

1. The system displays the information about the job and hides the *Place Bid* button and other admin/privileged actor's functionality (if any).

3c. The Actor is Admin

1. The system displays the information about the job
2. The system retrieves the list of placed bids
3. The system displays the list of *Placed Bids*
    a. The system displays a notification message `(NO_BIDS)` to inform the admin that there are no bids placed for the job yet.

3d. The Actor is the Job Owner and the Job's Status is Finished (Winners are chosen)

1. The system displays the information about the job
2. The system retrieves the list of winner Bids
3. The system displays the list of winner bids arranged by the Tradesman reputation
    a. The system displays a notification message `(NO_BIDS)` to inform the admin that there are no bids placed.

3f. The Actor is a Tradesman and has already placed a bid for the job

1. The system displays the information about the job and replaces the *Place Bid* button with *Edit Bid* and *Retract Bid* buttons.

3e. The Actor is the Job Owner and the Job's Status is Active or the Actor is Admin

1. The system displays the information about the job
2. The system displays buttons to edit and delete the job

Post New Job

Actor: Buyer

Lever: User Goal

Preconditions:

- The actor is logged in
- The actor has privileges to post jobs

Success Guarantee:

- A new Job Record is created

Main Success Scenario:

1. The actor arrives at a *Post New Job* page.
2. The system checks user privileges and displays *Post New Job Form*
3. The actor fills in the *Job Title*
4. The actor fills in the *Job Description*
5. The actor fills in the *Town*
6. The actor selects in the *County*
7. The actor selects the status of the *Planning Permission*
8. The actor selects the *Urgency* level
9. The actor selects the *Required Tradesmen* for the *Job*
10. The actor fills in the *Budget*
11. The actor includes *Attachments* that provide job description
12. The actor clicks the *Create New Job* button
13. The system checks the inputted information and creates a new Job Record.

Alternative Flows:

2a. The actor is not logged in

1. The system displays an error message (ACCT_REQ_POST_JOB) notifying that the actor needs to login or register to post a job.

2b. The actor doesn't have privileges to post a new job

1. The system displays an error message (PRIV_ERROR_POST_JOB) notifying that the actor cannot post new jobs.

11a. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 10

Edit Job

Actor: Buyer, Admin

Lever: User Goal, Admin Goal

Preconditions:

- The actor is logged in
- The actor has privileges to edit jobs
- The actor owns the job being edit or the action is Admin

Success Guarantee:

- The job record is updated
- A confirmation message is displayed

Main Success Scenario:

1. The actor clicks the *Edit Job* button
2. The system checks actor's privileges, retrieves the *Job Record* and displays *Edit Job Form*
3. The actor amends any number of fields and clicks the *Edit Job* button
4. The system checks the inputted information, checks actor's privileges, updates the *Job Record* and displays a confirmation message.

Alternative Flows:

2/4a. The actor is not logged in

2. The system displays an error message (ACCT_REQ_EDIT_JOB) notifying that the actor needs to login to edit the job.

2/4b. The actor doesn't have privileges to post a new job

2. The system displays an error message (NO_PRIV_EDIT_JOB) notifying that the actor cannot post new jobs.

2/4c. The job is finished

1. The system displays an error message (NO_PRIV_EDIT_JOB) notifying that the actor cannot post new jobs.

4d. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 3

Delete Job

Actor: Buyer, Admin

Lever: User Goal, Admin Goal

Preconditions:

- The actor is logged in
- The actor has privileges to delete Jobs

Success Guarantee:

- The Job is deleted and a confirmation message is displayed

Main Success Scenario:

1. The Actor is browsing the *Jobs List* page or *Full Job Description* page.
2. The Actor clicks on the *Delete Job* button
3. The system issues a confirmation dialog (`DELETE_CONF_JOB`)
4. The Actor confirms the action
5. The system checks actors permissions, deletes the job and displays the confirmation

Alternative Flows:

3a. The Actor has no privileges to delete the job (not admin or job owner)

1. The system displays an error message (NO_PRIV_DELETE_JOB) notifying that the actor that they do not have privileges to delete the Job.

4a. Reject confirmation

1. The actor rejects the confirmation (chooses not to delete the job)
2. The system hides the confirmation dialog and takes not further actions

5a. No Such Job Found

1. The system displays an error message (REQ_JOB_NOT_FOUND) to notify the actor that the requested job was not found.

Bids Use Cases

Place Bid

Actor: Tradesman

Lever: User Goal

Preconditions:

- The actor is logged in
- The actor has privileges to place bids
- The job is active
- The actor has not placed a bid for the job already (otherwise, the bid needs to be edited)

Success Guarantee:

- A new Bid record is created and a confirmation is displayed

Main Success Scenario:

1. The Actor is browsing the *Jobs List* page or *Full Job Description* page.
2. The Actor clicks on the *Place Bid* button
3. The system displays the *New Bid* dialog and asks for the bid *Value*
4. The Actor enters the bid value
5. The system checks actors privileges, creates a new *Bid* record, stores the Admin Rank, calculates the Indexed Bid Valued and displays a confirmation

Alternative Flows:

3a. The actor doesn't have privileges to place bids

1. The system displays an error message (ACCT_REQ_TO_PLACE_BIDS) to notify the actor that they need to login as a tradesman or create an account to place a bid.

3a. The Job is Finished

1. The system displays an error message (BID_JOB_OVER) to notify the actor that they cannot place a bid for a finished job

3b. The actor already placed a bid for the job

1. The system displays an error message (BID_ALREADY_EXISTS) to notify the actor that they cannot place several bids for a job and need to edit their existing bid.

5a. One or several validation rules did not pass

2. The system highlights the fields that failed validation
3. The actor corrects the errors
4. The actor resumes from step 5

Edit a Bid

Actor: Tradesman, Admin

Lever: User Goal, Admin Goal

Preconditions:

- The actor is logged in
- The actor has privileges to edit bids
- The job is active

Success Guarantee:

- The Bid record is edited and a confirmation is displayed

Main Success Scenario:

1. The Actor is browsing the *Jobs List* page or *Full Job Description* page.
2. The Actor clicks on the *Edit Bid* button
3. The system displays the *Edit Bid* dialog and asks for the new bid *Value*
4. The Actor enters the new bid value
5. The system checks actors privileges, edits the *Bid Value* , recalculates the *Indexed Bid Value* and displays a confirmation

Alternative Flows:

3a. The actor doesn't have privileges to edit bids

1. The system displays an error message (NO_PRIV_EDIT_BID) to notify the actor that they do not have sufficient privileges to edit the bid.

3b. The Bid is not found

1. The system displays an error message (REQ_BID_NOT_FOUND) to notify the actor that the bid they tried to edit was not found.

3c. The Job is Finished

2. The system displays an error message (BID_JOB_OVER) to notify the actor that they cannot place a bid for a finished job

5a. One or several validation rules did not pass

1. The system highlights the fields that failed validation
2. The actor corrects the errors
3. The actor resumes from step 5

Retract a Bid

Actor: Tradesman, Admin

Lever: User Goal, Admin Goal

Preconditions:

- The actor is logged in
- The actor has privileges to retract bids
- The job is active
- The actor (Tradesman) has a bid for the job in question

Success Guarantee:

- The Bid record is deleted and a confirmation message is displayed

Main Success Scenario:

1. The Actor is browsing the *Jobs List* page or *Full Job Description* page.
2. The Actor clicks on the *Retract Bid* button
3. The system retrieves the bid and displays the *Retract Bid* confirmation dialog
4. The Actor confirms the action
5. The system checks actors privileges, retrieves the bid, ensures the job is still active, deletes the *Bid* record, updates tradesman profile decrementing bids made count and displays a confirmation

Alternative Flows:

3/5a. The actor doesn't have privileges to delete bids

1. The system displays an error message (NO_PRIV_DELETE_BID) to notify the actor that they do not have sufficient privileges to delete the bid.

3/5b. The Bid is not found

1. The system displays an error message (REQ_BID_NOT_FOUND) to notify the actor that the bid they tried to delete was not found.

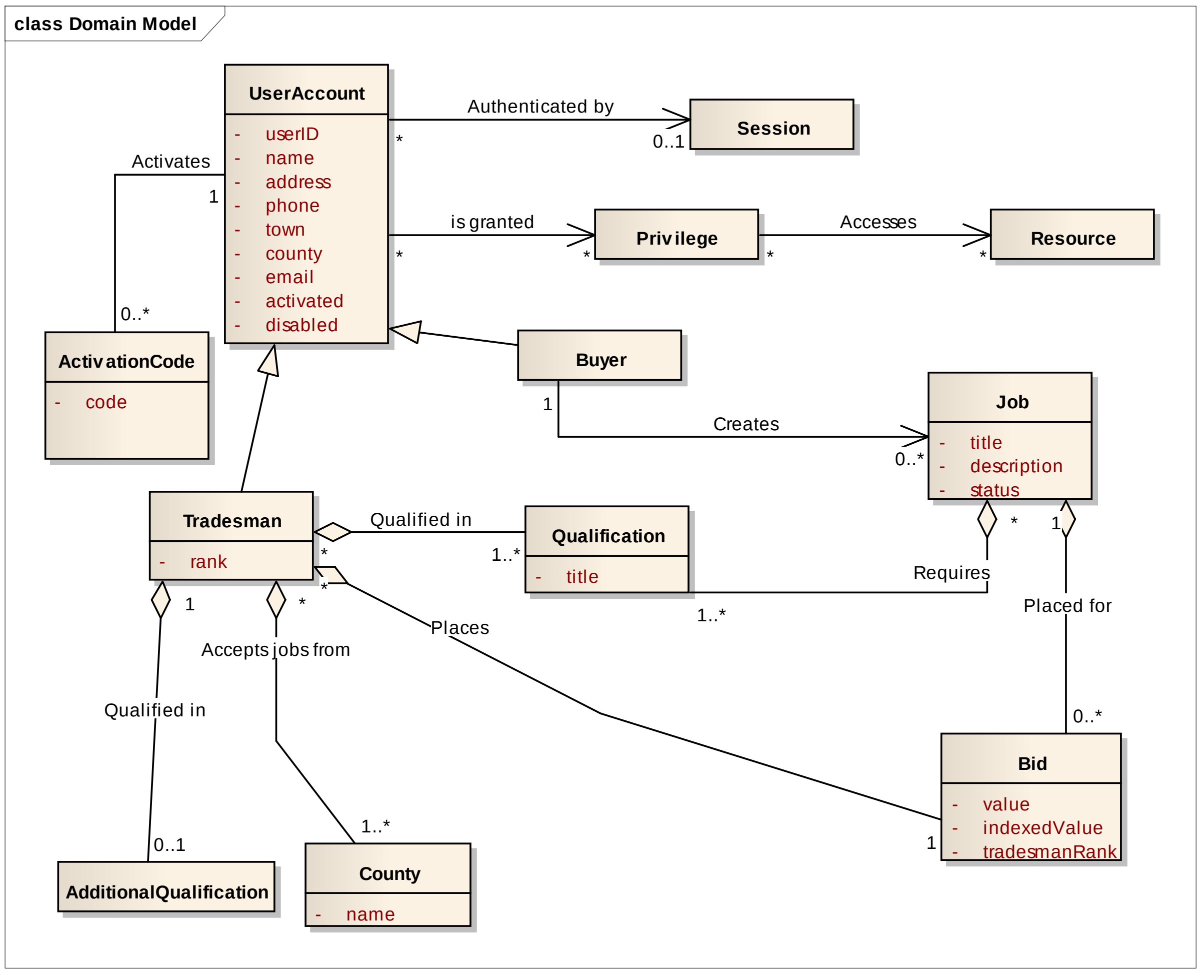
3/5c. The Job is Finished

1. The system displays an error message (BID_JOB_OVER) to notify the actor that they cannot delete the bid for a finished job

Domain Model

The process of extraction business objects (domain objects) from all of the uses cases above and determining the relationships between them produces the domain model provided on the diagram below.

The domain model represents an overview of the system in terms of the objects it contains and how they relate to each other both in the direction of relationships and their quantifications.

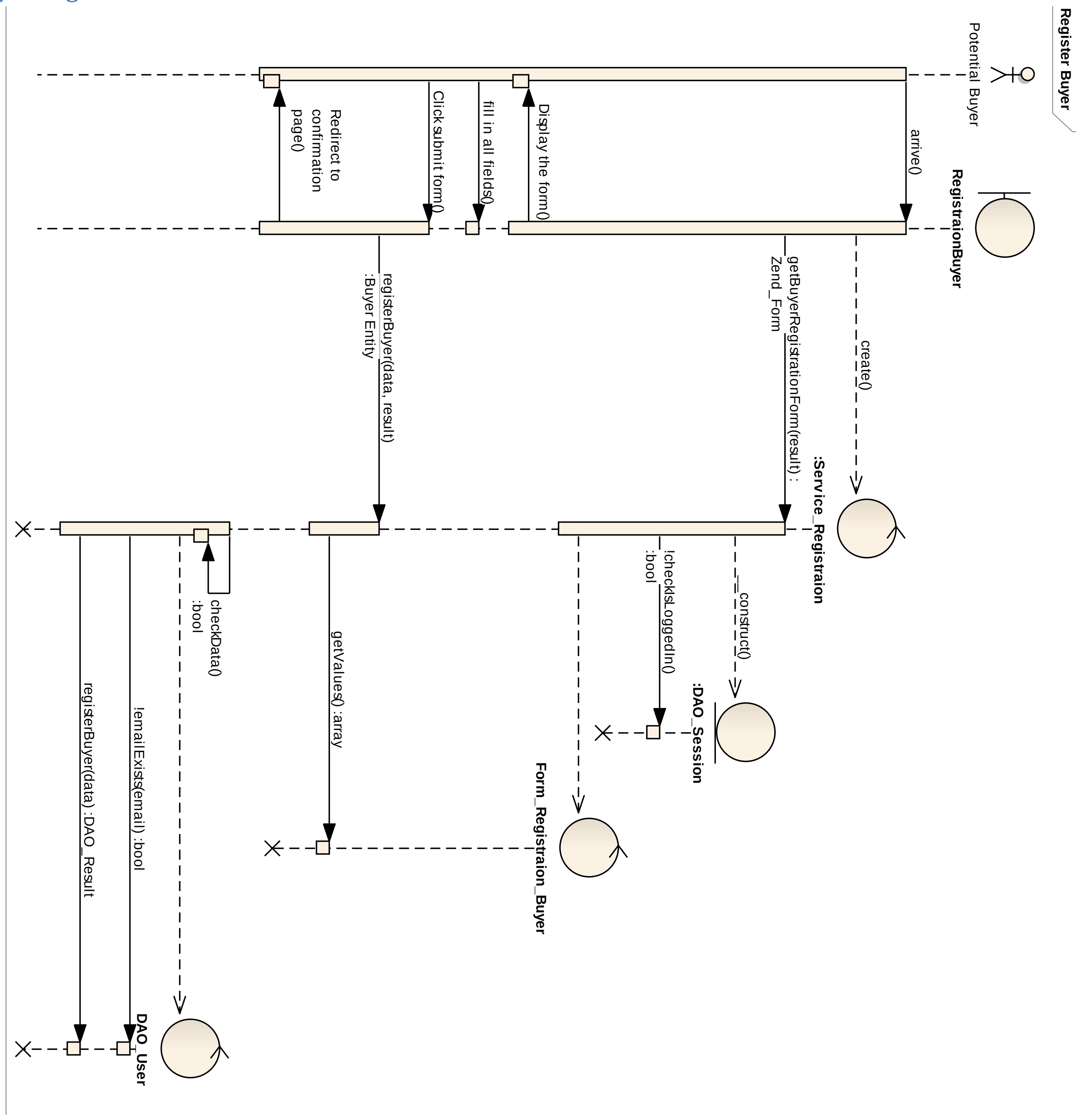


Sequence Diagrams

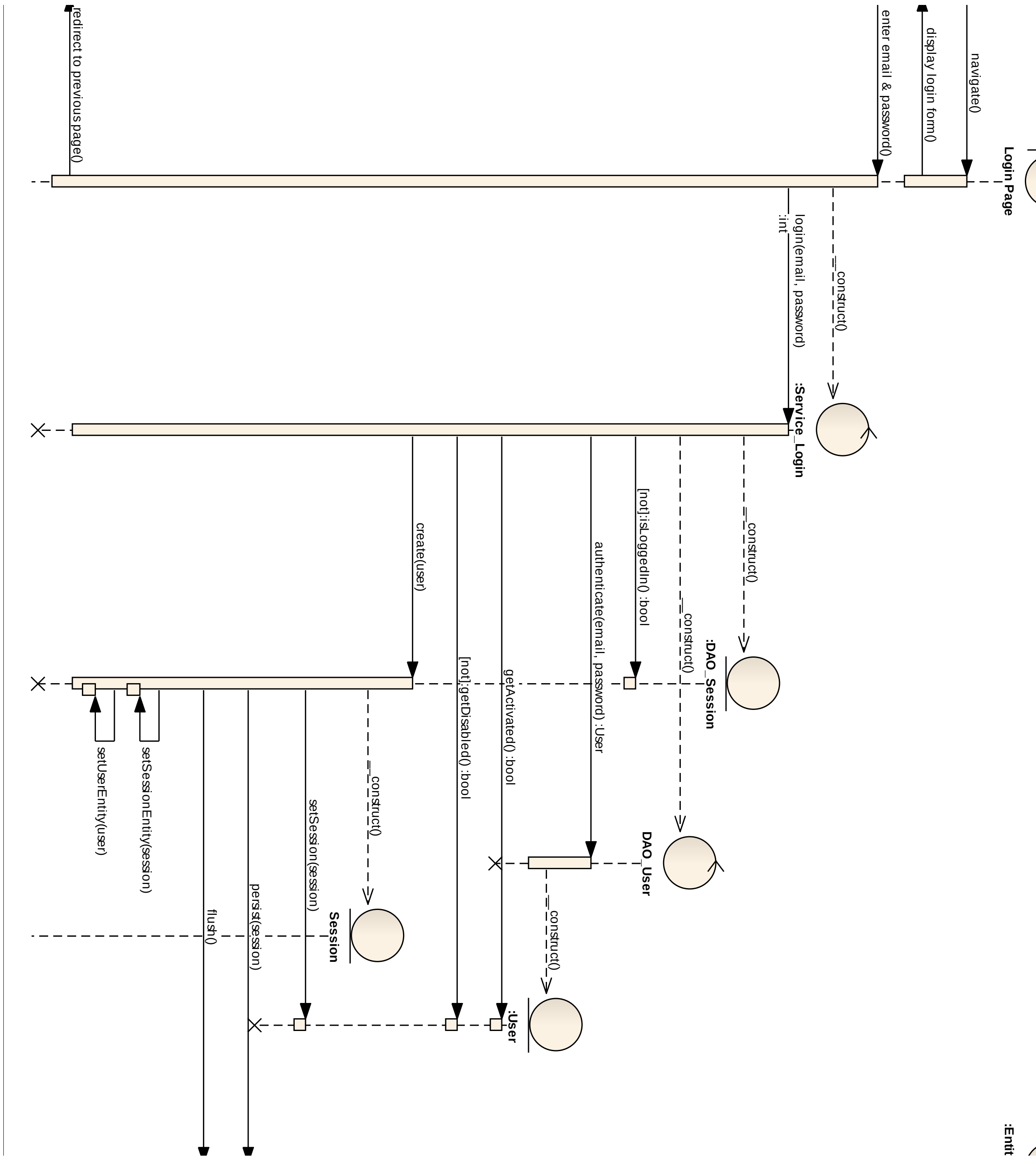
The final step before we can produce a detailed class diagram is to roughly model, for each of the use cases, the interaction between the objects in our domain model and to introduce other classes for each of the layers in the architecture. While producing the interaction diagram, it is necessary to do to rough sketches of the classes and refine them with each new sequence diagram.

Accounts System

Buyer Registration

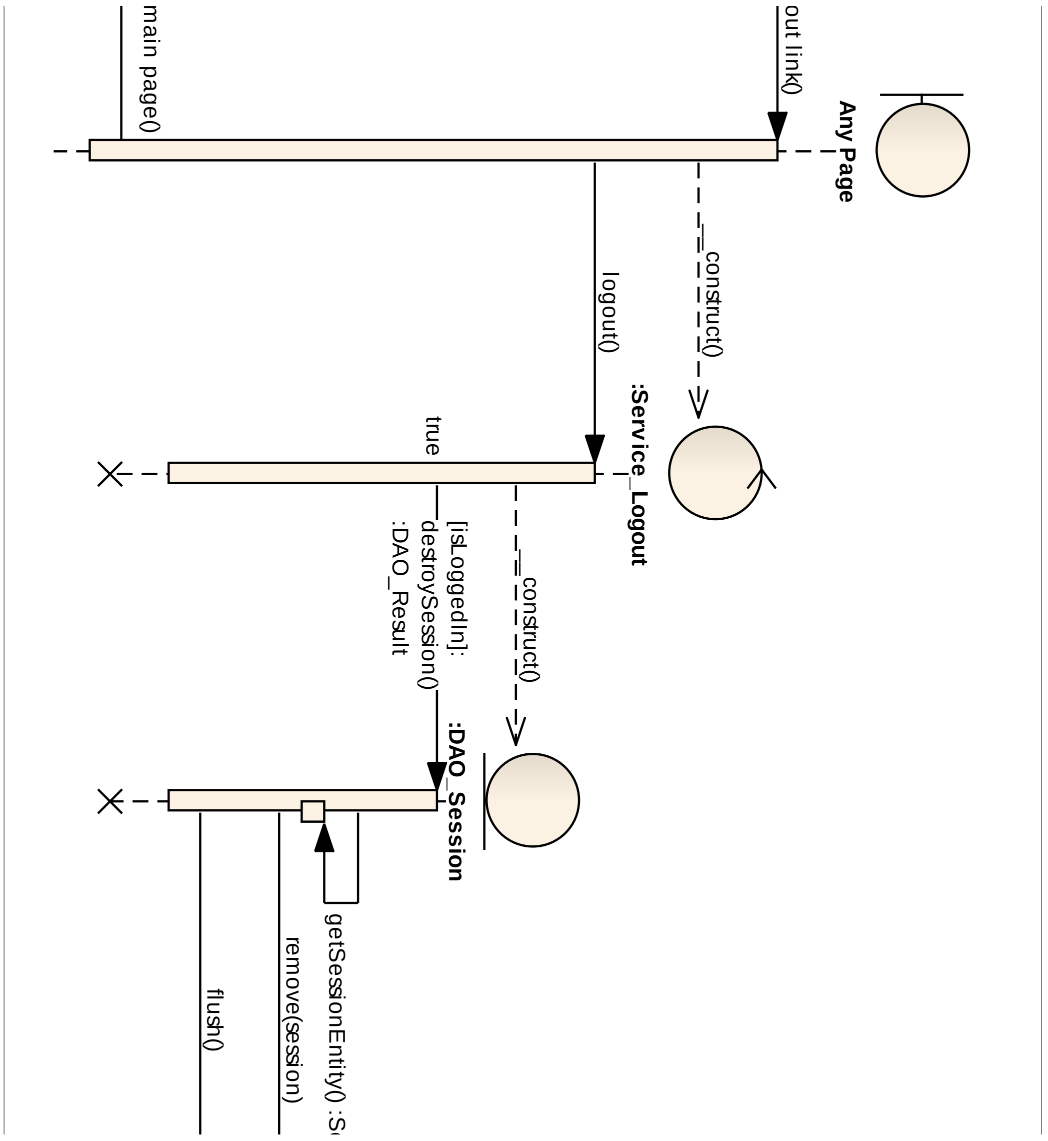


Login

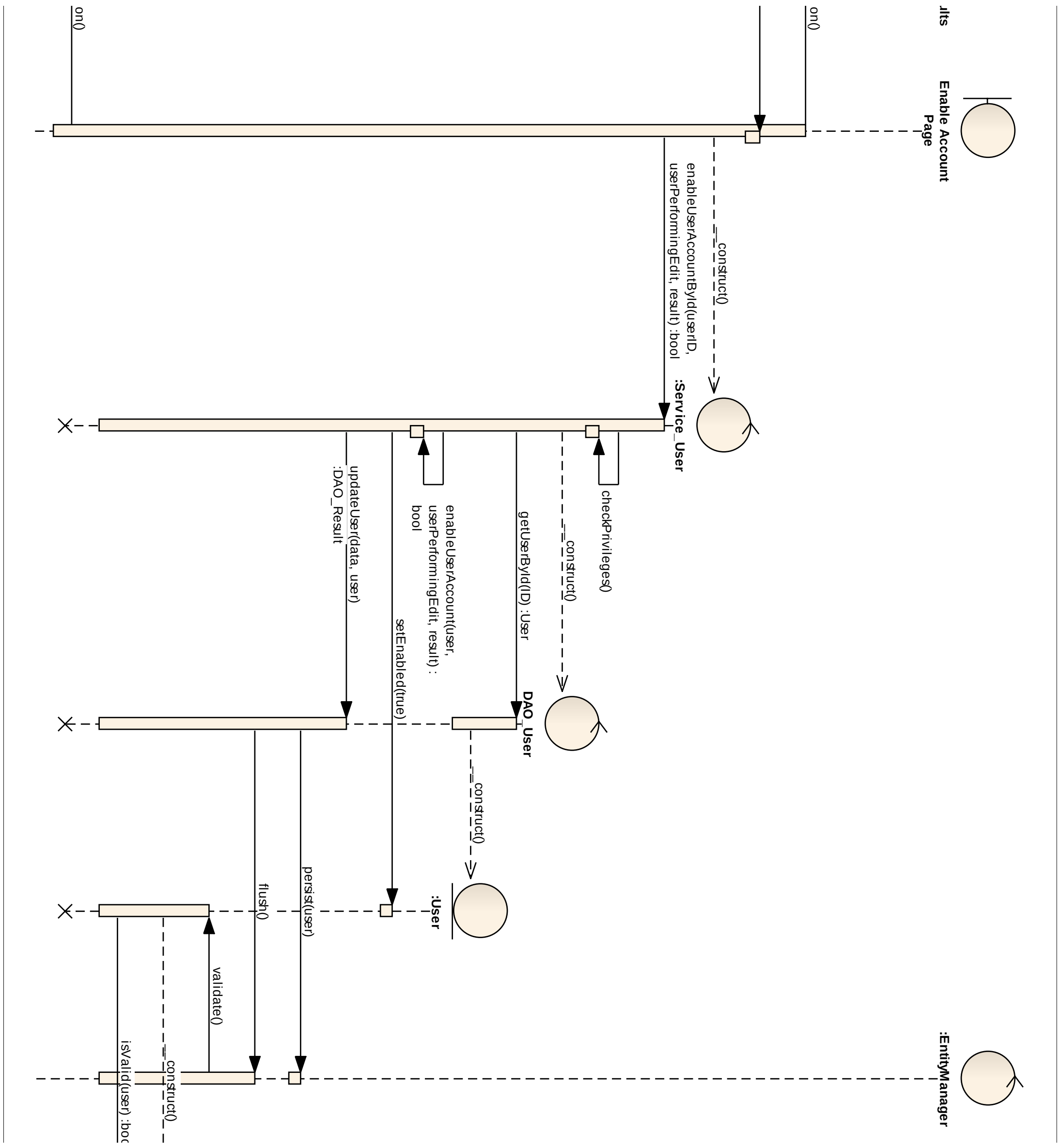


Login Page
navigate()
display login form()
enter email & password()
redirect to previous page()
__construct()
login(email, password) :int
:Service_Login
__construct()
[not]:isLoggedIn() :bool
__construct()
authenticate(email, password) :User
getActivated() :bool
[not]:getDisabled() :bool
create(user)
:DAO_Session
DAO_User
__construct()
__construct()
setSession(session)
persist(session)
flush()
setSessionEntity(session)
setUserEntity(user)
Session
:User
:Enti

Logout

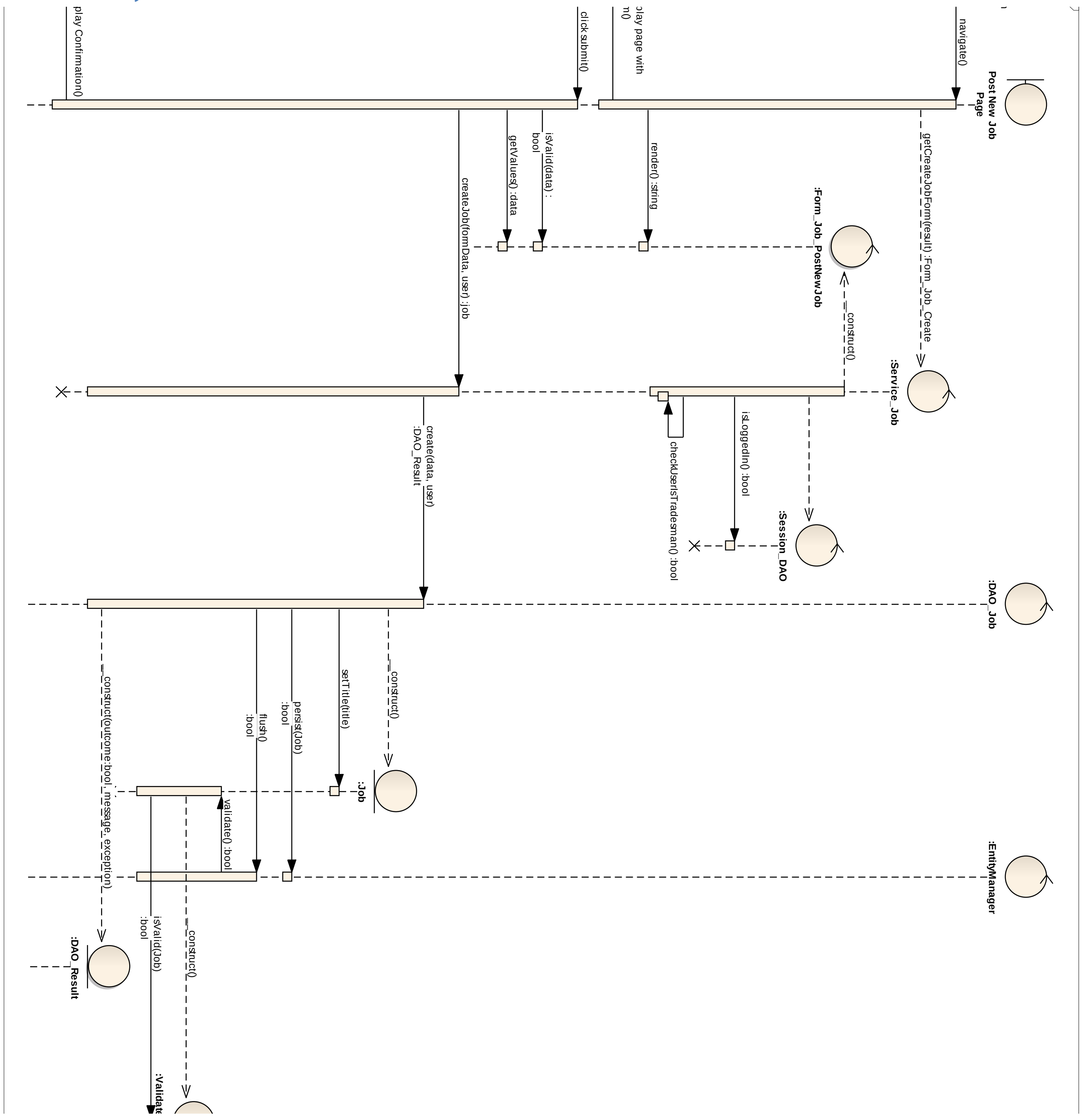

Any Page
:Service_Logout
:DAO_Session
out link()
main page()
__construct()
logout()
true
[isLoggedIn]: destroySession() :DAO_Result
__construct()
getSessionEntity() :S
remove(session)
flush()

Enable Account

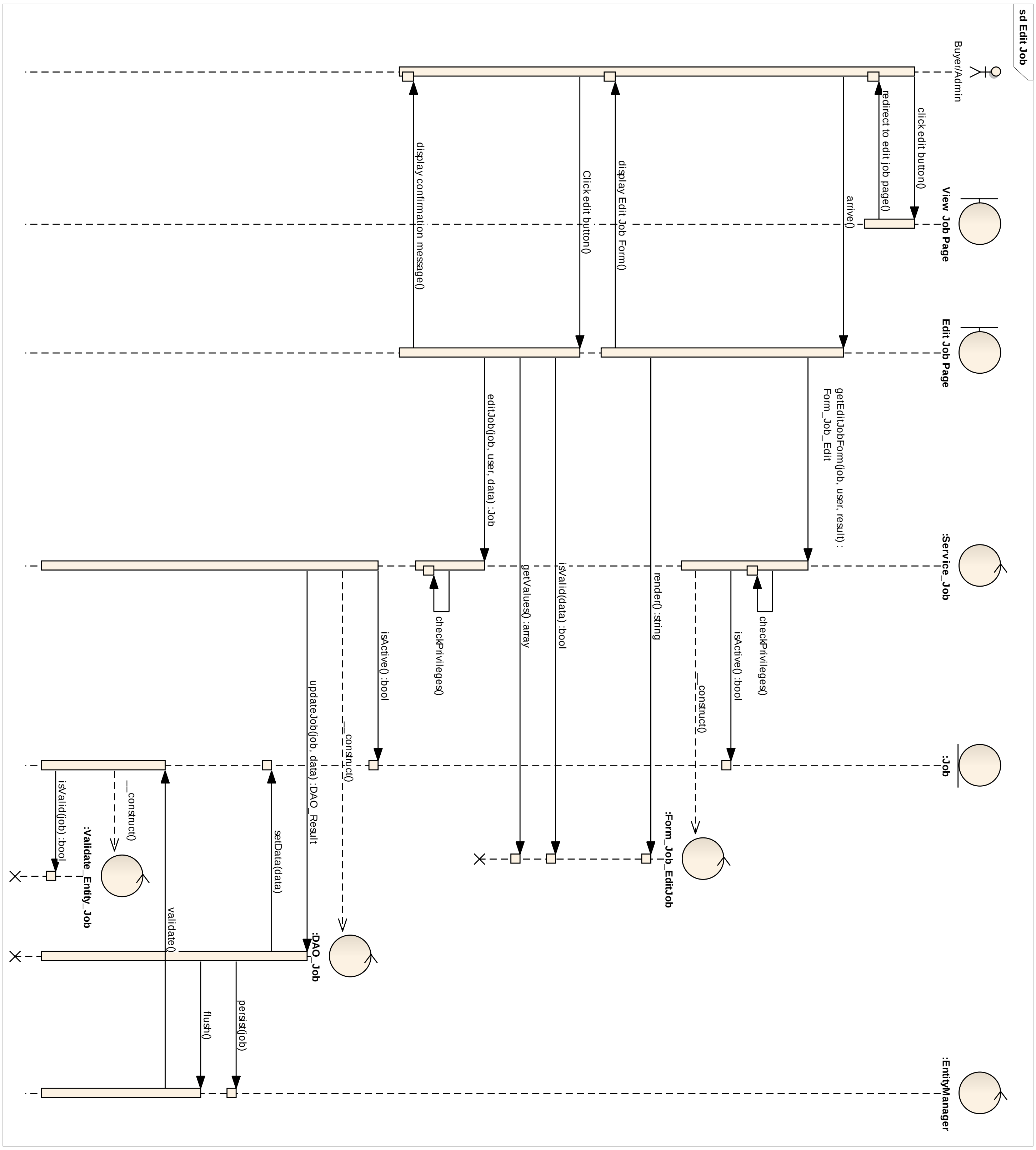


Jobs System

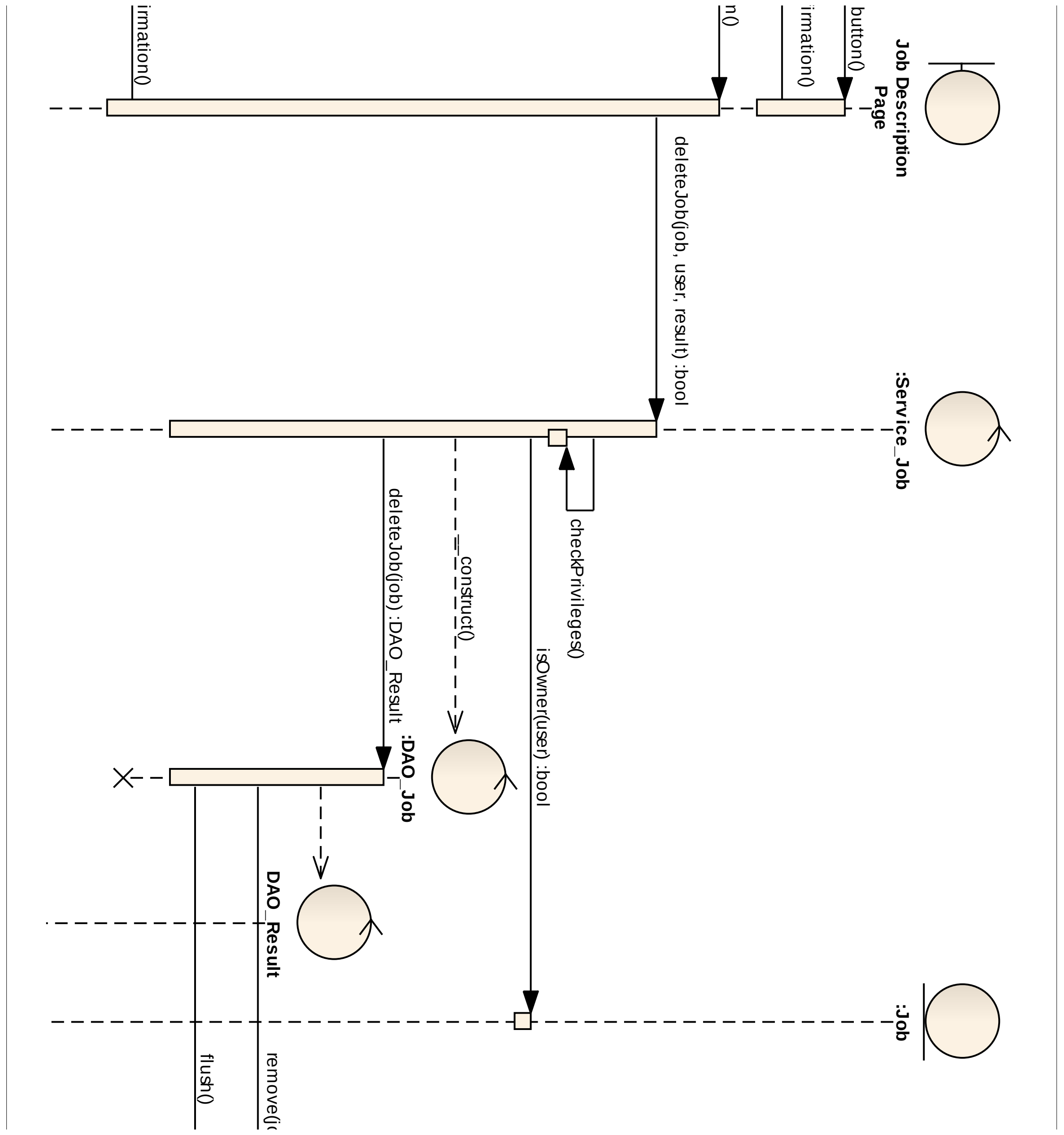
Post New Job



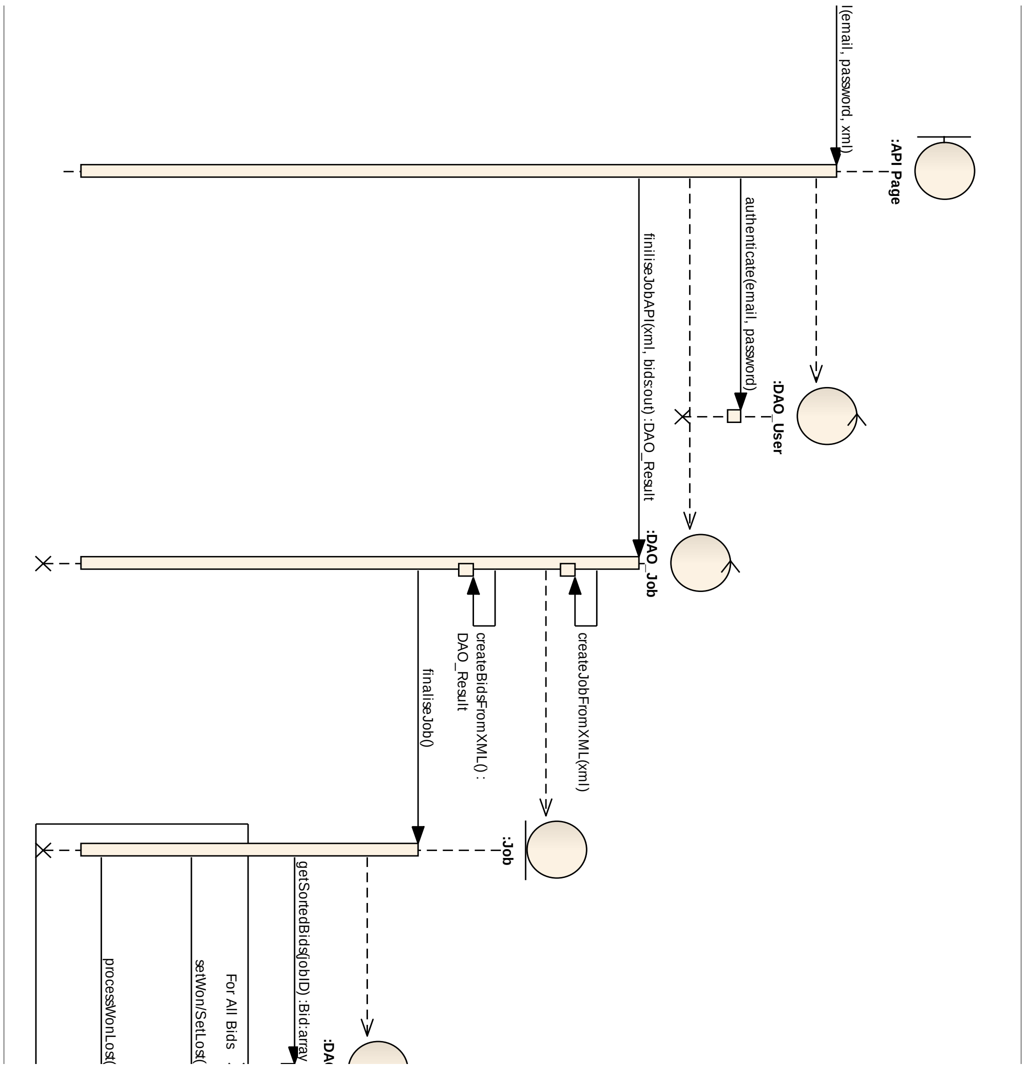
Edit Job


sd Edit Job
Buyer/Admin
View Job Page
Edit Job Page
:Service_Job
:Job
:Form_Job_EditJob
:Validate_Entity_Job
:DAO_Job
:EntityManager
click edit button()
redirect to edit job page()
arrive()
getEditJobForm(job, user, result) :
Form_Job_Edit
checkPrivileges()
isActive() :bool
__construct()
render() :string
display Edit Job Form()
Click edit button()
editJob(job, user, data) :Job
checkPrivileges()
isValid(data) :bool
getValues() :array
isActive() :bool
__construct()
updateJob(job, data) :DAO_Result
setData(data)
__construct()
isValid(job) :bool
validate()
persist(job)
flush()
display confirmation message()

Delete Job

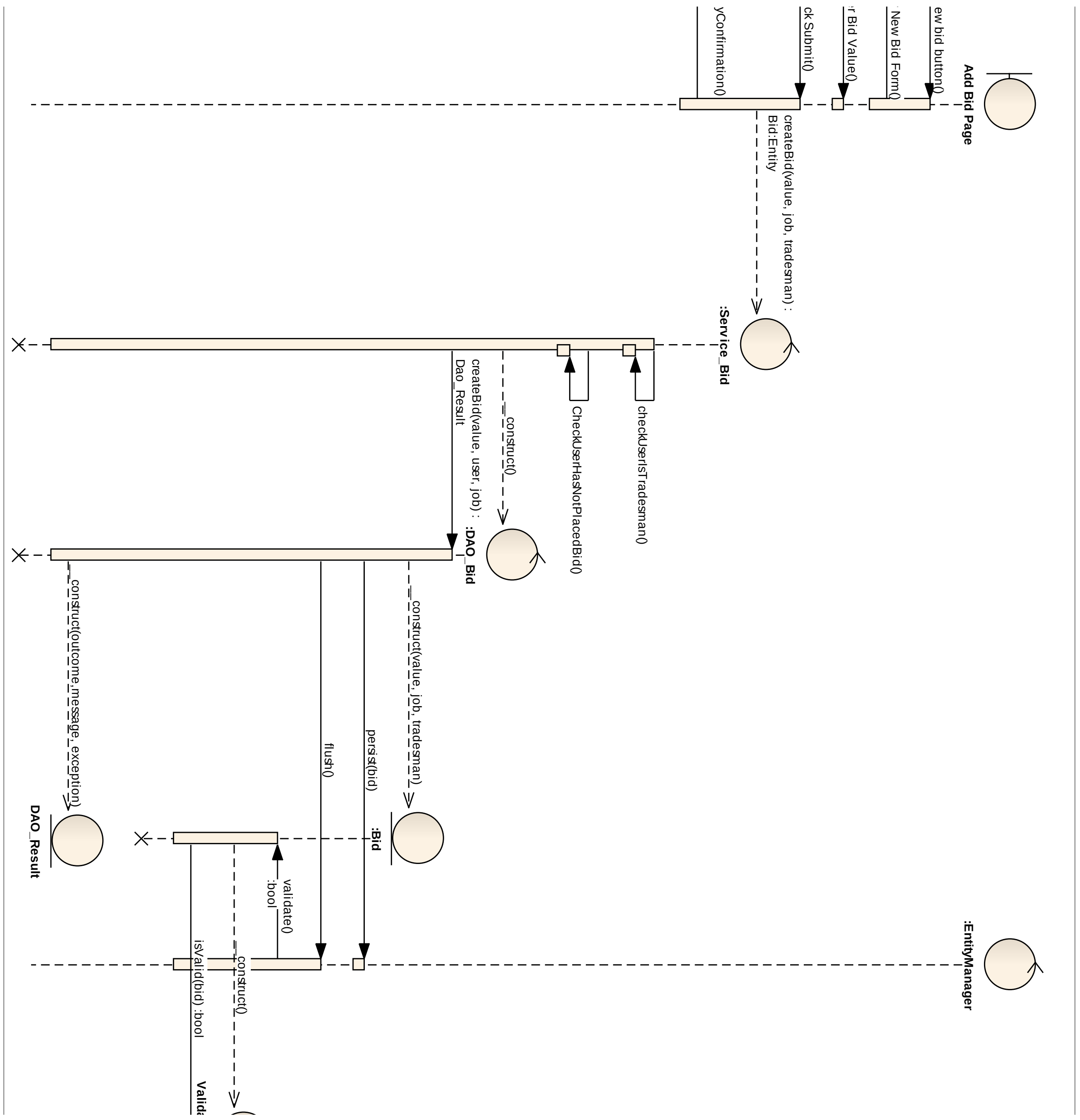

Job Description Page
:Service_Job
:Job
:DAO_Job
DAO_Result
button()
irmation()
n()
irmation()
deleteJob(job, user, result) :bool
checkPrivileges()
isOwner(user) :bool
__construct()
deleteJob(job) :DAO_Result
remove(j
flush()

Finalise the Job

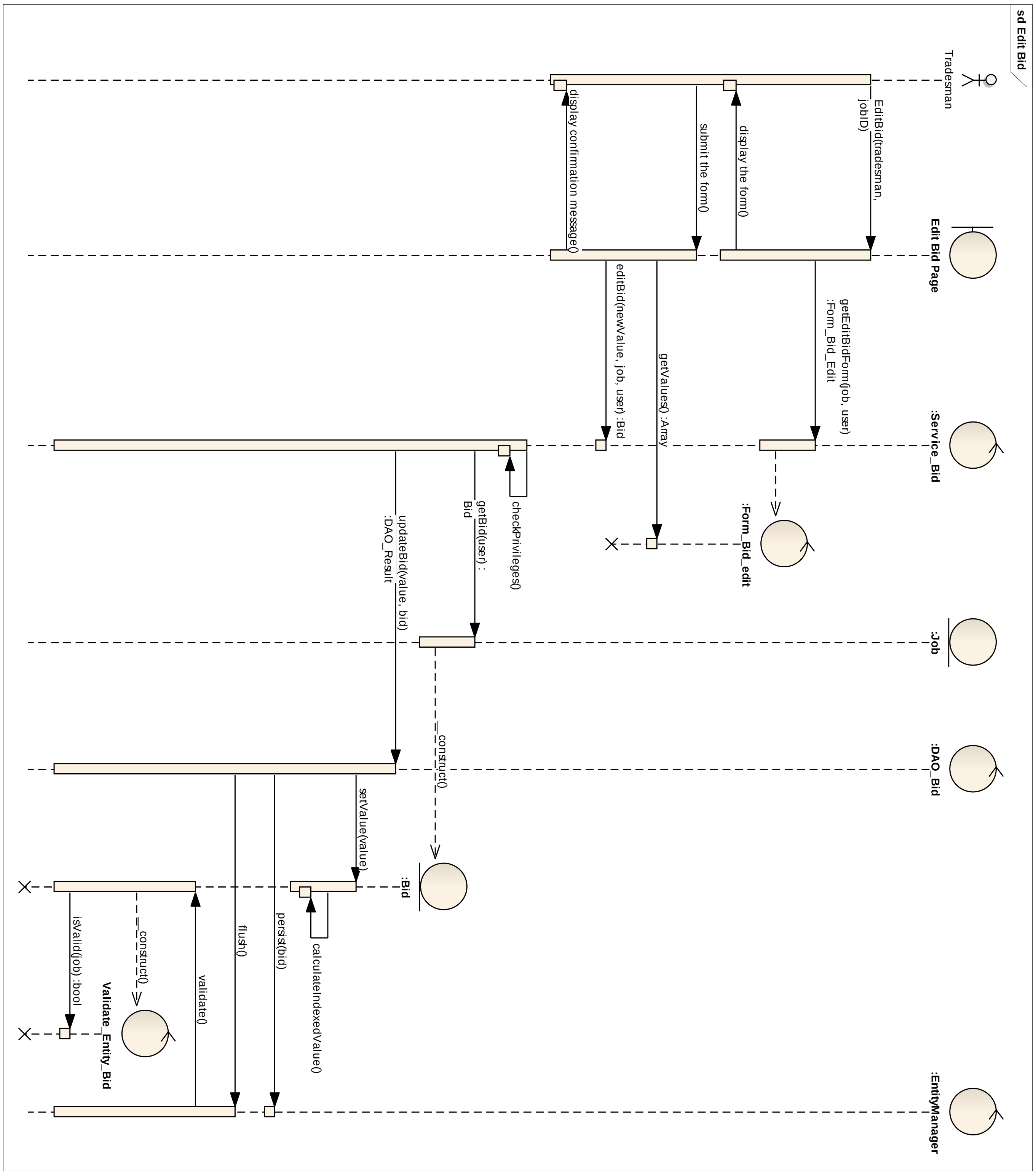

l(email, password, xml)
:API Page
authenticate(email, password)
:DAO_User
finiliseJobAPI(xml, bidsout) :DAO_Result
:DAO_Job
createJobFromXML(xml)
createBidsFromXML() : DAO_Result
finaliseJob()
:Job
getSortedBids(jobID) :Bid:array
For All Bids
setWon/SetLost(
processWonLost(

Bids System

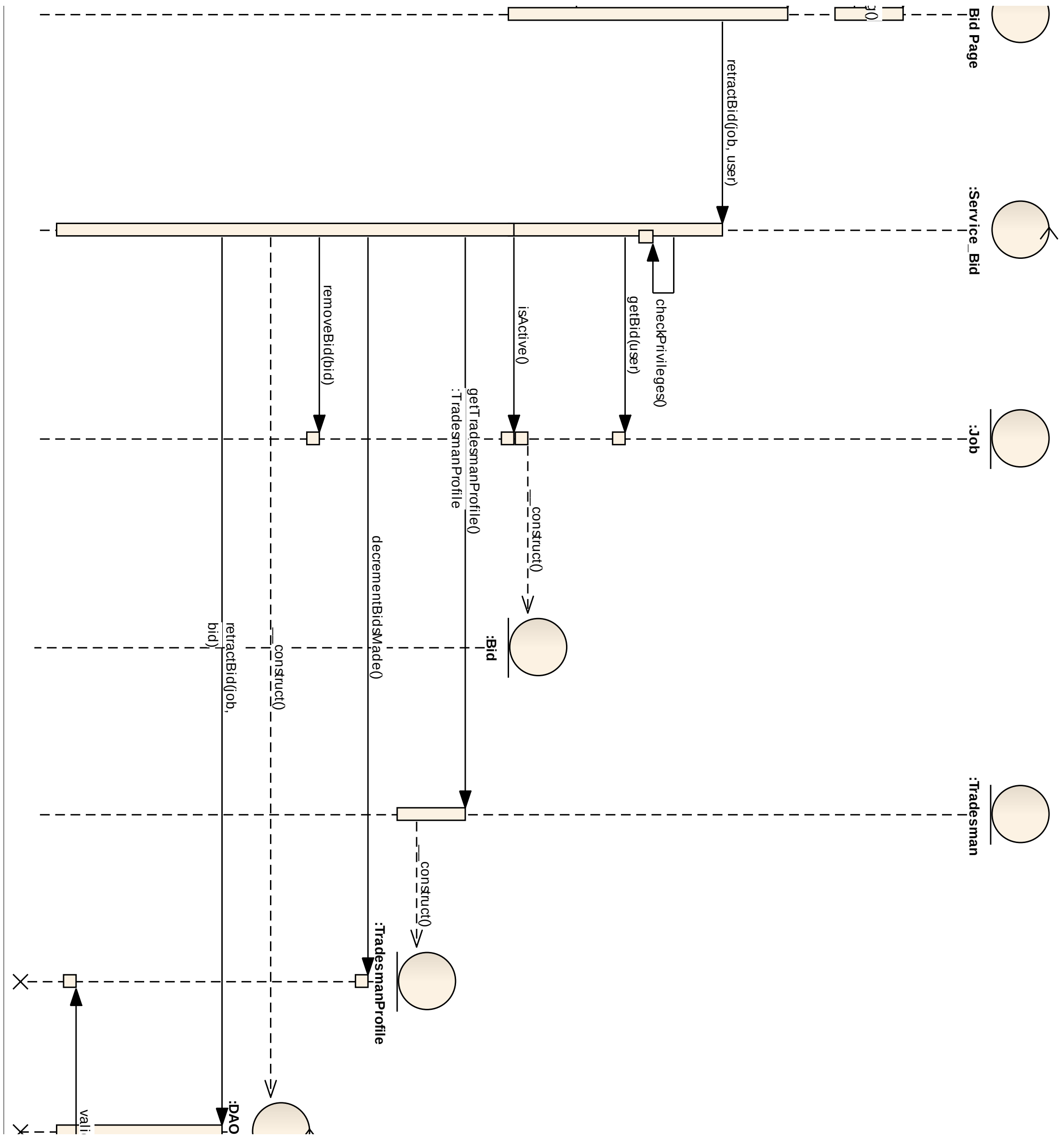
Create Bid



Edit Bid


sd Edit Bid
Tradesman
Edit Bid Page
:Service_Bid
:Form_Bid_edit
:Job
:DAO_Bid
:Bid
Validate_Entity_Bid
:EntityManager
EditBid(tradesman, jobID)
display the form()
submit the form()
display confirmation message()
getEditBidForm(job, user) :Fom_Bid_Edit
getValues() :Array
editBid(newValue, job, user) :Bid
checkPrivileges()
getBid(user) : Bid
updateBid(value, bid) :DAO_Result
__construct()
setValue(value)
calculateIndexedValue()
persist(bid)
flush()
validate()
__construct()
isValid(job) :bool

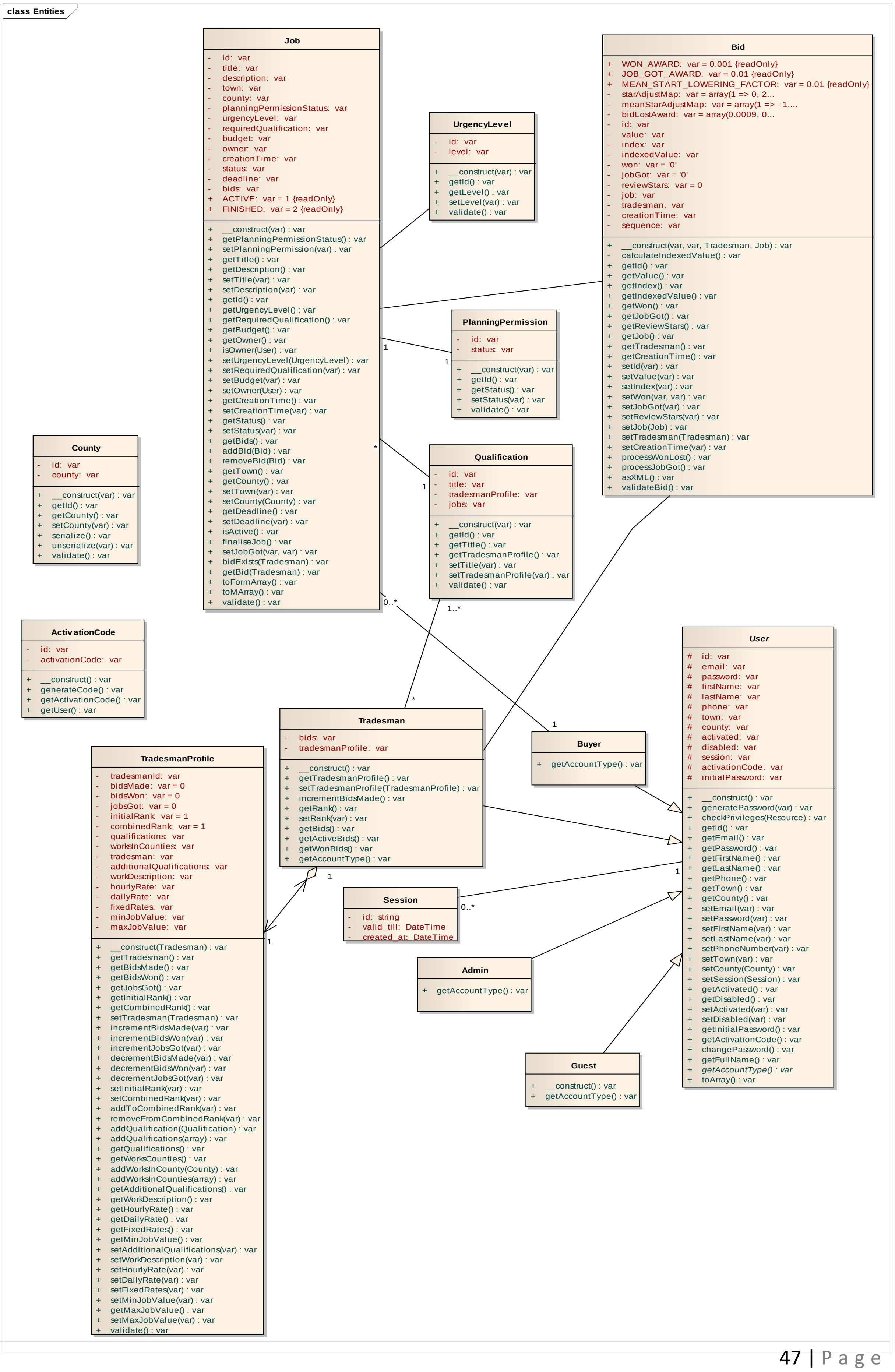
Delete Bid



Class Diagram

Entities

Entities are domain objects from the domain model transferred into proper entity classes with defined and implemented functionality. Entities encapsulate and group data and functionality in them about particular concept of the system. Entities are operated upon by the application logic layer (services and DAO) that modifies entities, transparently persists, updates and retrieves them to and from the Datastore (MySQL).

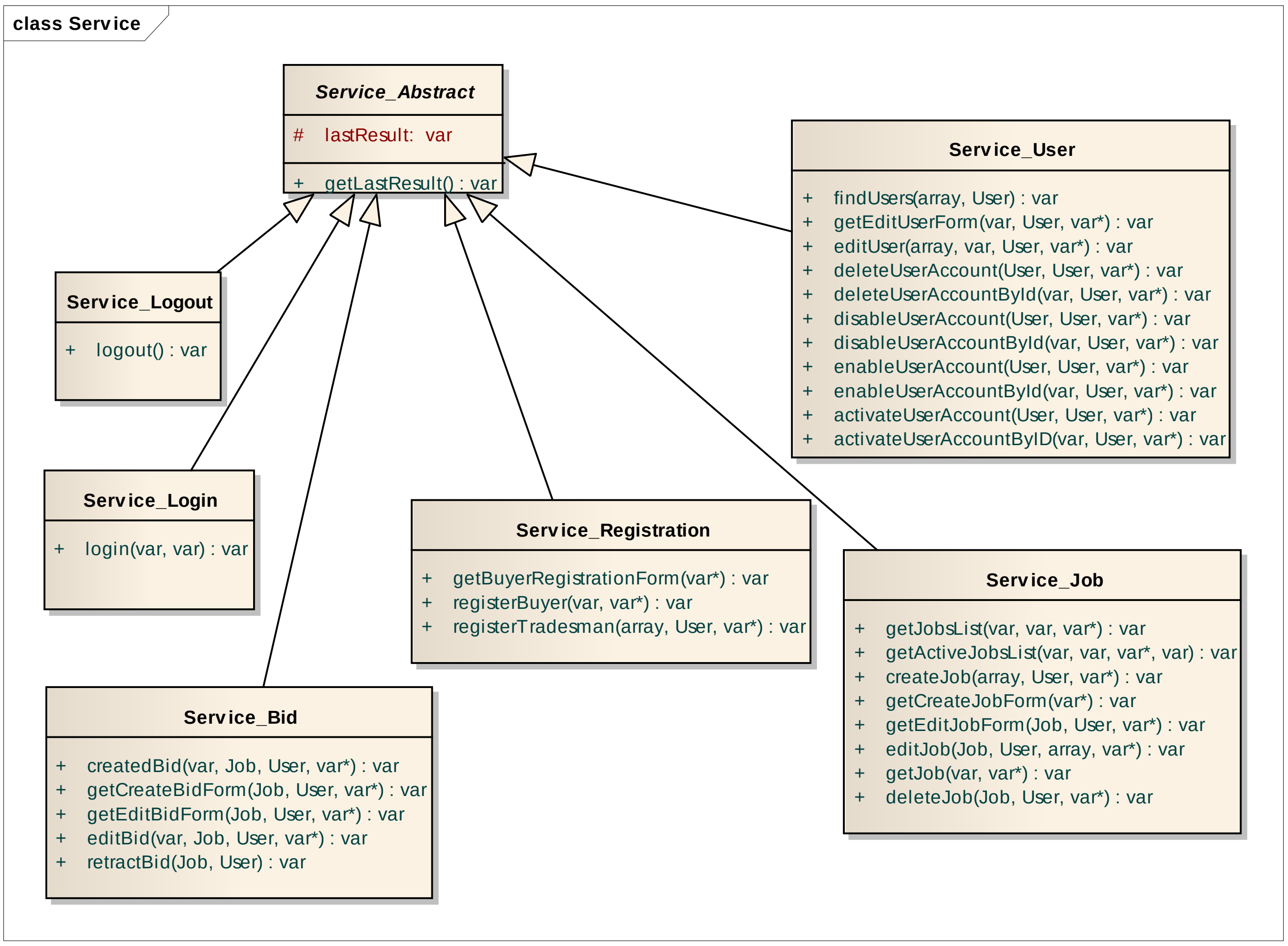


class Entities
Job
- id: var
- title: var
- description: var
- town: var
- county: var
- planningPermissionStatus: var
- urgencyLevel: var
- requiredQualification: var
- budget: var
- owner: var
- creationTime: var
- status: var
- deadline: var
- bids: var
+ ACTIVE: var = 1 {readOnly}
+ FINISHED: var = 2 {readOnly}
+ __construct(var) : var
+ getPlanningPermissionStatus() : var
+ setPlanningPermission(var) : var
+ getTitle() : var
+ getDescription() : var
+ setTitle(var) : var
+ setDescription(var) : var
+ getId() : var
+ getUrgencyLevel() : var
+ getRequiredQualification() : var
+ getBudget() : var
+ getOwner() : var
+ isOwner(User) : var
+ setUrgencyLevel(UrgencyLevel) : var
+ setRequiredQualification(var) : var
+ setBudget(var) : var
+ setOwner(User) : var
+ getCreationTime() : var
+ setCreationTime(var) : var
+ getStatus() : var
+ setStatus(var) : var
+ getBids() : var
+ addBid(Bid) : var
+ removeBid(Bid) : var
+ getTown() : var
+ getCounty() : var
+ setTown(var) : var
+ setCounty(County) : var
+ getDeadline() : var
+ setDeadline(var) : var
+ isActive() : var
+ finaliseJob() : var
+ setJobGot(var, var) : var
+ bidExists(Tradesman) : var
+ getBid(Tradesman) : var
+ toFormArray() : var
+ toMArray() : var
+ validate() : var
UrgencyLevel
- id: var
- level: var
+ __construct(var) : var
+ getId() : var
+ getLevel() : var
+ setLevel(var) : var
+ validate() : var
Bid
+ WON_AWARD: var = 0.001 {readOnly}
+ JOB_GOT_AWARD: var = 0.01 {readOnly}
+ MEAN_START_LOWERING_FACTOR: var = 0.01 {readOnly}
- starAdjustMap: var = array(1 => 0, 2...
- meanStarAdjustMap: var = array(1 => - 1....
- bidLostAward: var = array(0.0009, 0...
- id: var
- value: var
- index: var
- indexedValue: var
- won: var = '0'
- jobGot: var = '0'
- reviewStars: var = 0
- job: var
- tradesman: var
- creationTime: var
- sequence: var
+ __construct(var, var, Tradesman, Job) : var
- calculateIndexedValue() : var
+ getId() : var
+ getValue() : var
+ getIndex() : var
+ getIndexedValue() : var
+ getWon() : var
+ getJobGot() : var
+ getReviewStars() : var
+ getJob() : var
+ getTradesman() : var
+ getCreationTime() : var
+ setId(var) : var
+ setValue(var) : var
+ setIndex(var) : var
+ setWon(var, var) : var
+ setJobGot(var) : var
+ setReviewStars(var) : var
+ setJob(Job) : var
+ setTradesman(Tradesman) : var
+ setCreationTime(var) : var
+ processWonLost() : var
+ processJobGot() : var
+ asXML() : var
+ validateBid() : var
PlanningPermission
- id: var
- status: var
+ __construct(var) : var
+ getId() : var
+ getStatus() : var
+ setStatus(var) : var
+ validate() : var
1
1
*
1
Qualification
- id: var
- title: var
- tradesmanProfile: var
- jobs: var
+ __construct(var) : var
+ getId() : var
+ getTitle() : var
+ getTradesmanProfile() : var
+ setTitle(var) : var
+ setTradesmanProfile(var) : var
+ validate() : var
County
- id: var
- county: var
+ __construct(var) : var
+ getId() : var
+ getCounty() : var
+ setCounty(var) : var
+ serialize() : var
+ unserialize(var) : var
+ validate() : var
ActivationCode
- id: var
- activationCode: var
+ __construct() : var
+ generateCode() : var
+ getActivationCode() : var
+ getUser() : var
0..*
1..*
*
1
User
# id: var
# email: var
# password: var
# firstName: var
# lastName: var
# phone: var
# town: var
# county: var
# activated: var
# disabled: var
# session: var
# activationCode: var
# initialPassword: var
+ __construct() : var
+ generatePassword(var) : var
+ checkPrivileges(Resource) : var
+ getId() : var
+ getEmail() : var
+ getPassword() : var
+ getFirstName() : var
+ getLastName() : var
+ getPhone() : var
+ getTown() : var
+ getCounty() : var
+ setEmail(var) : var
+ setPassword(var) : var
+ setFirstName(var) : var
+ setLastName(var) : var
+ setPhoneNumber(var) : var
+ setTown(var) : var
+ setCounty(County) : var
+ setSession(Session) : var
+ getActivated() : var
+ getDisabled() : var
+ setActivated(var) : var
+ setDisabled(var) : var
+ getInitialPassword() : var
+ getActivationCode() : var
+ changePassword() : var
+ getFullName() : var
+ getAccountType() : var
+ toArray() : var
Tradesman
- bids: var
- tradesmanProfile: var
+ __construct() : var
+ getTradesmanProfile() : var
+ setTradesmanProfile(TradesmanProfile) : var
+ incrementBidsMade() : var
+ getRank() : var
+ setRank(var) : var
+ getBids() : var
+ getActiveBids() : var
+ getWonBids() : var
+ getAccountType() : var
Buyer
+ getAccountType() : var
TradesmanProfile
- tradesmanId: var
- bidsMade: var = 0
- bidsWon: var = 0
- jobsGot: var = 0
- initialRank: var = 1
- combinedRank: var = 1
- qualifications: var
- worksInCounties: var
- tradesman: var
- additionalQualifications: var
- workDescription: var
- hourlyRate: var
- dailyRate: var
- fixedRates: var
- minJobValue: var
- maxJobValue: var
+ __construct(Tradesman) : var
+ getTradesman() : var
+ getBidsMade() : var
+ getBidsWon() : var
+ getJobsGot() : var
+ getInitialRank() : var
+ getCombinedRank() : var
+ setTradesman(Tradesman) : var
+ incrementBidsMade(var) : var
+ incrementBidsWon(var) : var
+ incrementJobsGot(var) : var
+ decrementBidsMade(var) : var
+ decrementBidsWon(var) : var
+ decrementJobsGot(var) : var
+ setInitialRank(var) : var
+ setCombinedRank(var) : var
+ addToCombinedRank(var) : var
+ removeFromCombinedRank(var) : var
+ addQualification(Qualification) : var
+ addQualifications(array) : var
+ getQualifications() : var
+ getWorksCounties() : var
+ addWorksInCounty(County) : var
+ addWorksInCounties(array) : var
+ getAdditionalQualifications() : var
+ getWorkDescription() : var
+ getHourlyRate() : var
+ getDailyRate() : var
+ getFixedRates() : var
+ getMinJobValue() : var
+ setAdditionalQualifications(var) : var
+ setWorkDescription(var) : var
+ setHourlyRate(var) : var
+ setDailyRate(var) : var
+ setFixedRates(var) : var
+ setMinJobValue(var) : var
+ getMaxJobValue() : var
+ setMaxJobValue(var) : var
+ validate() : var
1
1
Session
- id: string
- valid_till: DateTime
- created_at: DateTime
0..*
1
Admin
+ getAccountType() : var
Guest
+ __construct() : var
+ getAccountType() : var

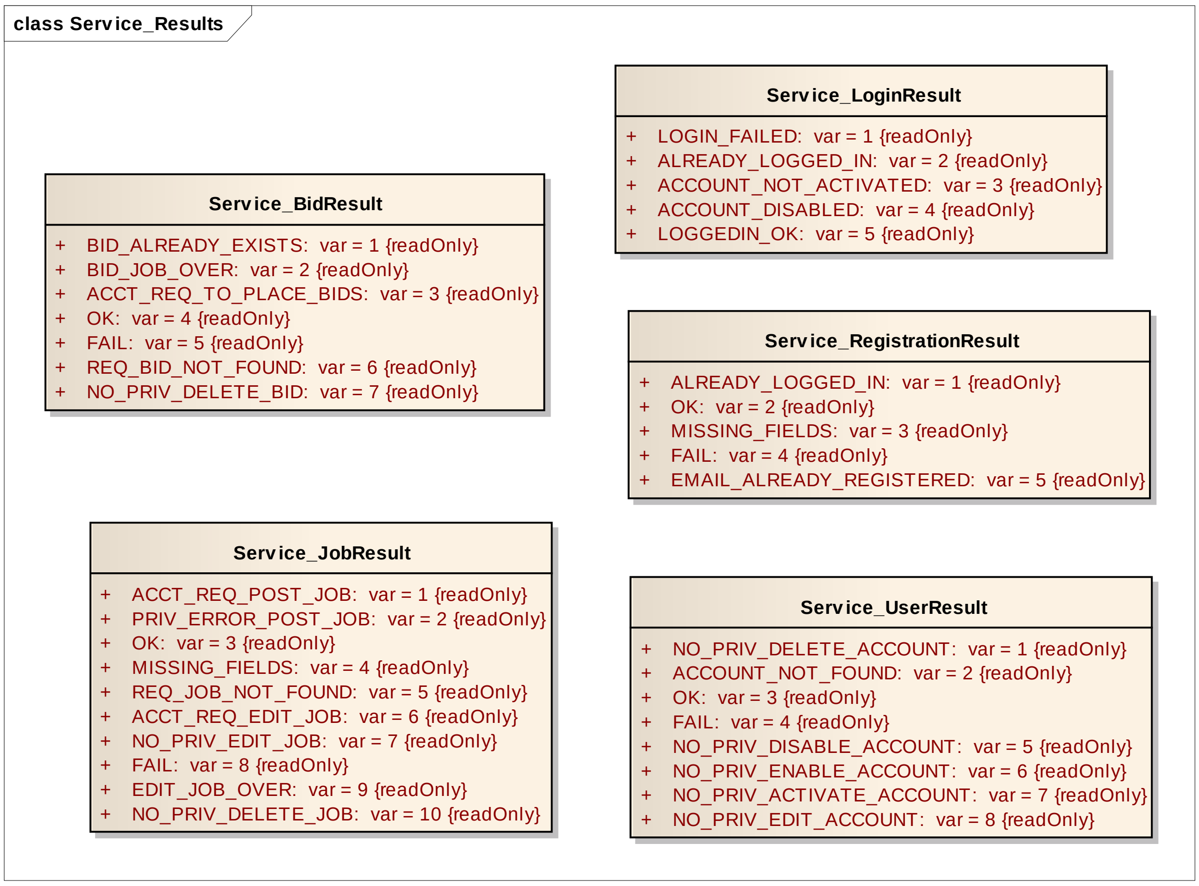
Services

Application services contain all the logic of the application. Essentially, they are the API of the application. They carry out all tasks within the application, control access to information, fetch data from data stores by interfacing to other layers of the system and provide a result to the presentation layers of the application (controllers). They make decisions about what sort of action and result to carry out in a particular case.

The services are directly mapped to uses cases and account for each step in every use case. Services return well defined results for any operation which are documented in uses cases and further specified in the "Notifications and Error messages" table in the appendix.

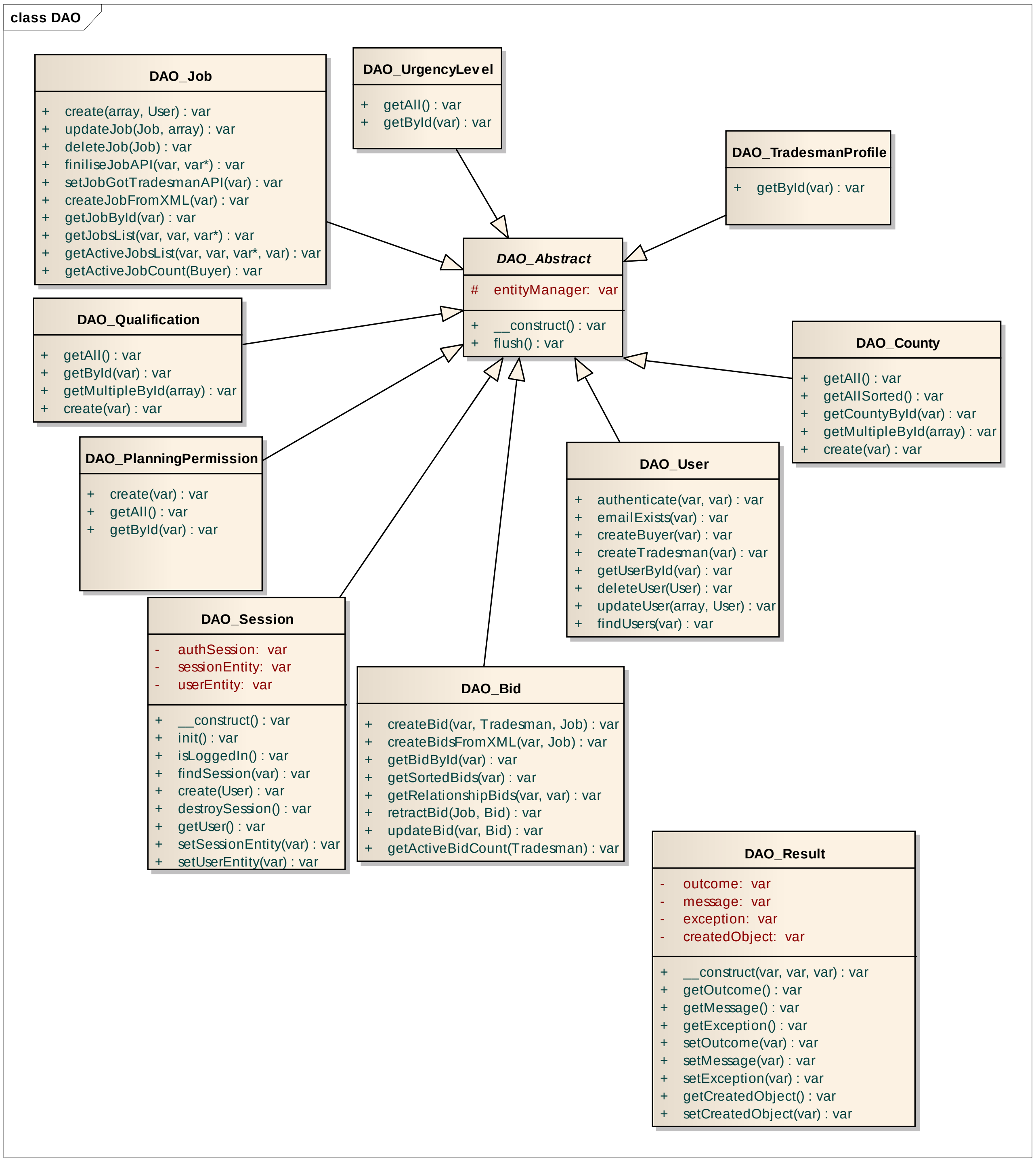

class Service
Service_Abstract
# lastResult: var
+ getLastResult() : var
Service_User
+ findUsers(array, User) : var
+ getEditUserForm(var, User, var*) : var
+ editUser(array, var, User, var*) : var
+ deleteUserAccount(User, User, var*) : var
+ deleteUserAccountById(var, User, var*) : var
+ disableUserAccount(User, User, var*) : var
+ disableUserAccountById(var, User, var*) : var
+ enableUserAccount(User, User, var*) : var
+ enableUserAccountById(var, User, var*) : var
+ activateUserAccount(User, User, var*) : var
+ activateUserAccountByID(var, User, var*) : var
Service_Logout
+ logout() : var
Service_Login
+ login(var, var) : var
Service_Registration
+ getBuyerRegistrationForm(var*) : var
+ registerBuyer(var, var*) : var
+ registerTradesman(array, User, var*) : var
Service_Job
+ getJobsList(var, var, var*) : var
+ getActiveJobsList(var, var, var*, var) : var
+ createJob(array, User, var*) : var
+ getCreateJobForm(var*) : var
+ getEditJobForm(Job, User, var*) : var
+ editJob(Job, User, array, var*) : var
+ getJob(var, var*) : var
+ deleteJob(Job, User, var*) : var
Service_Bid
+ createdBid(var, Job, User, var*) : var
+ getCreateBidForm(Job, User, var*) : var
+ getEditBidForm(Job, User, var*) : var
+ editBid(var, Job, User, var*) : var
+ retractBid(Job, User) : var

Results classes below are helper classes for services. They specify what sort of a result a service can return. The API consumer then, after a call to an API method in a service can check the result of the operation which is a finite enumerated set.


class Service_Results
Service_LoginResult
+ LOGIN_FAILED: var = 1 {readOnly}
+ ALREADY_LOGGED_IN: var = 2 {readOnly}
+ ACCOUNT_NOT_ACTIVATED: var = 3 {readOnly}
+ ACCOUNT_DISABLED: var = 4 {readOnly}
+ LOGGEDIN_OK: var = 5 {readOnly}
Service_BidResult
+ BID_ALREADY_EXISTS: var = 1 {readOnly}
+ BID_JOB_OVER: var = 2 {readOnly}
+ ACCT_REQ_TO_PLACE_BIDS: var = 3 {readOnly}
+ OK: var = 4 {readOnly}
+ FAIL: var = 5 {readOnly}
+ REQ_BID_NOT_FOUND: var = 6 {readOnly}
+ NO_PRIV_DELETE_BID: var = 7 {readOnly}
Service_RegistrationResult
+ ALREADY_LOGGED_IN: var = 1 {readOnly}
+ OK: var = 2 {readOnly}
+ MISSING_FIELDS: var = 3 {readOnly}
+ FAIL: var = 4 {readOnly}
+ EMAIL_ALREADY_REGISTERED: var = 5 {readOnly}
Service_JobResult
+ ACCT_REQ_POST_JOB: var = 1 {readOnly}
+ PRIV_ERROR_POST_JOB: var = 2 {readOnly}
+ OK: var = 3 {readOnly}
+ MISSING_FIELDS: var = 4 {readOnly}
+ REQ_JOB_NOT_FOUND: var = 5 {readOnly}
+ ACCT_REQ_EDIT_JOB: var = 6 {readOnly}
+ NO_PRIV_EDIT_JOB: var = 7 {readOnly}
+ FAIL: var = 8 {readOnly}
+ EDIT_JOB_OVER: var = 9 {readOnly}
+ NO_PRIV_DELETE_JOB: var = 10 {readOnly}
Service_UserResult
+ NO_PRIV_DELETE_ACCOUNT: var = 1 {readOnly}
+ ACCOUNT_NOT_FOUND: var = 2 {readOnly}
+ OK: var = 3 {readOnly}
+ FAIL: var = 4 {readOnly}
+ NO_PRIV_DISABLE_ACCOUNT: var = 5 {readOnly}
+ NO_PRIV_ENABLE_ACCOUNT: var = 6 {readOnly}
+ NO_PRIV_ACTIVATE_ACCOUNT: var = 7 {readOnly}
+ NO_PRIV_EDIT_ACCOUNT: var = 8 {readOnly}

Data Access Objects

Data access objects contain very little logic. Their main task is to take in entities and persist them to the Datastore and return the result of the operation (DAO_Result) back to the service layer.



Filters

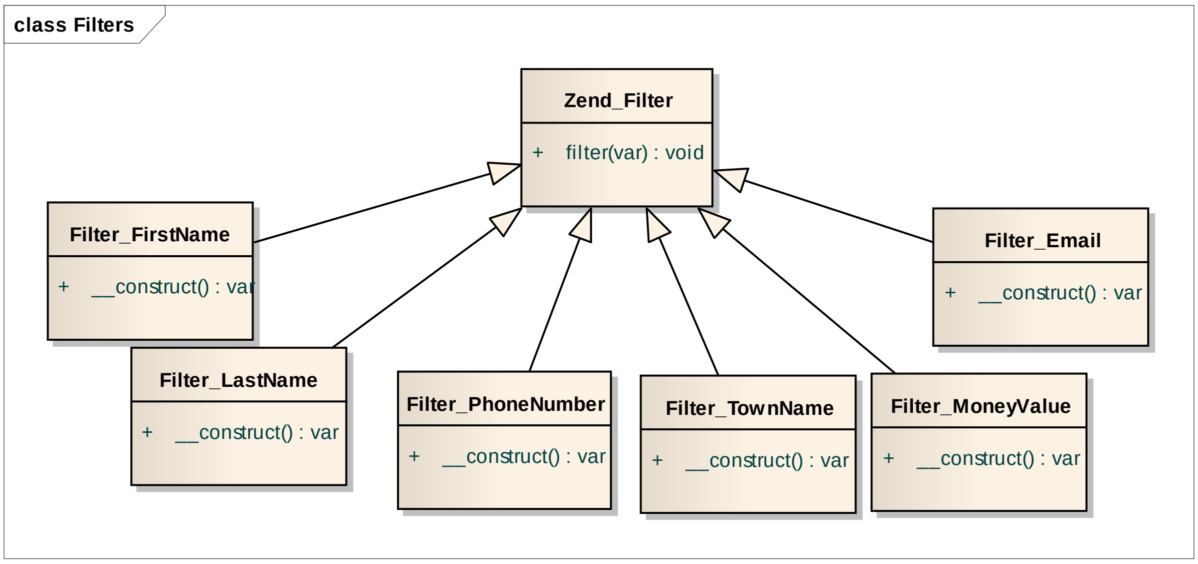
Filters are classes which ensure that the data that comes from the user is properly filtered: all potentially dangerous statements are removed that can lead to SQL injections, XSS attacks and other threats. Filters also attempt to bring data to required format by stripping out and type forcing data to a desired format.

Filters are used in forms (detailed below) and in entities upon two events:

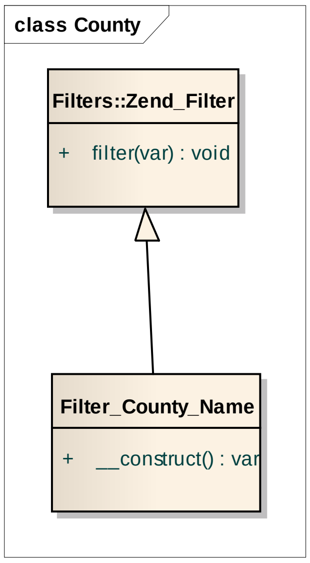
1. Before a newly created entity is persisted to the database
2. Before an existing entity is resynchronised (updated) to the database

After the filters are executed, validators are run to ensure that filtered data conforms to a required format (provided in the functional specification). If validation fails, the user is provided with a meaningful error message (again, documented in the functional specification) for all the input entries that did not pass the validation.

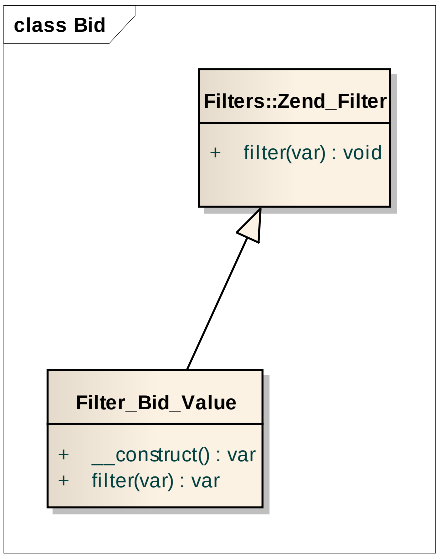
Generic



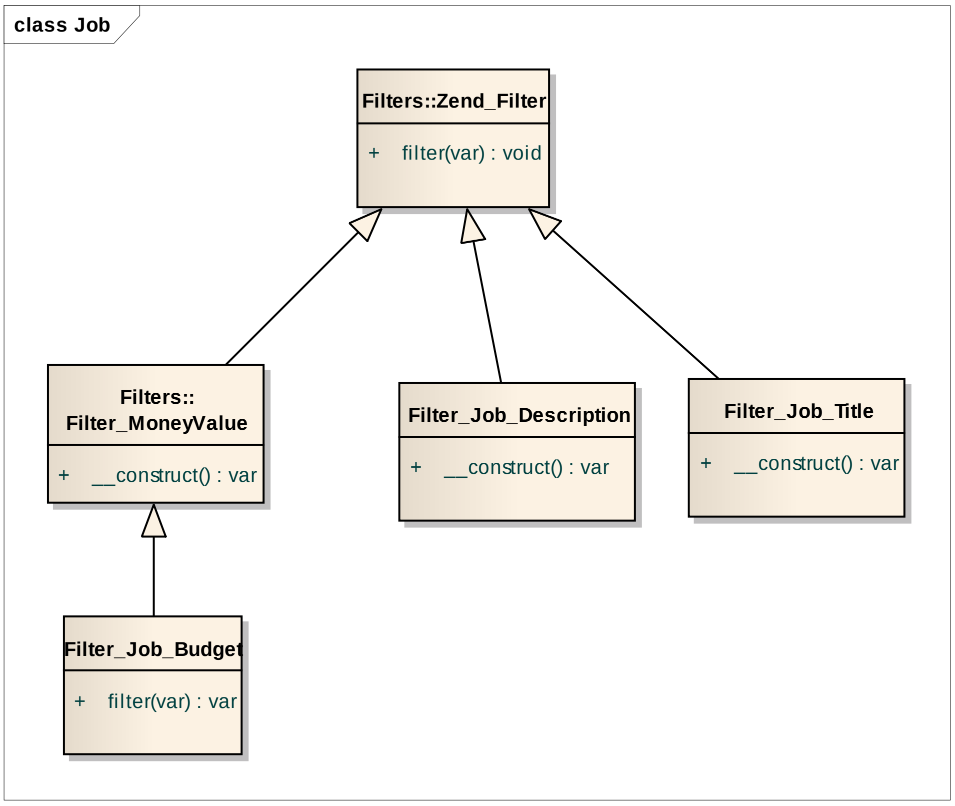
County


class County
Filters::Zend_Filter
+ filter(var) : void
Filter_County_Name
+ __construct() : var

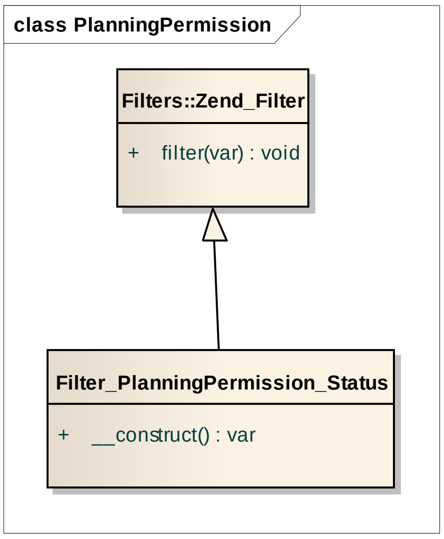
Bid


class Bid
Filters::Zend_Filter
+ filter(var) : void
Filter_Bid_Value
+ __construct() : var
+ filter(var) : var

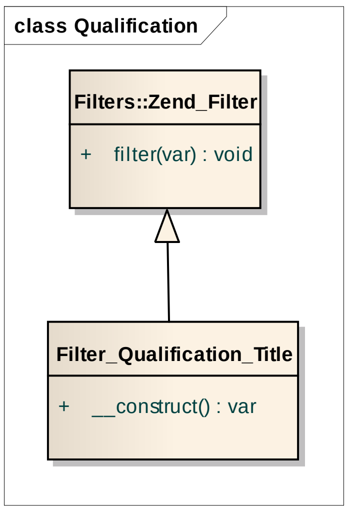
Job



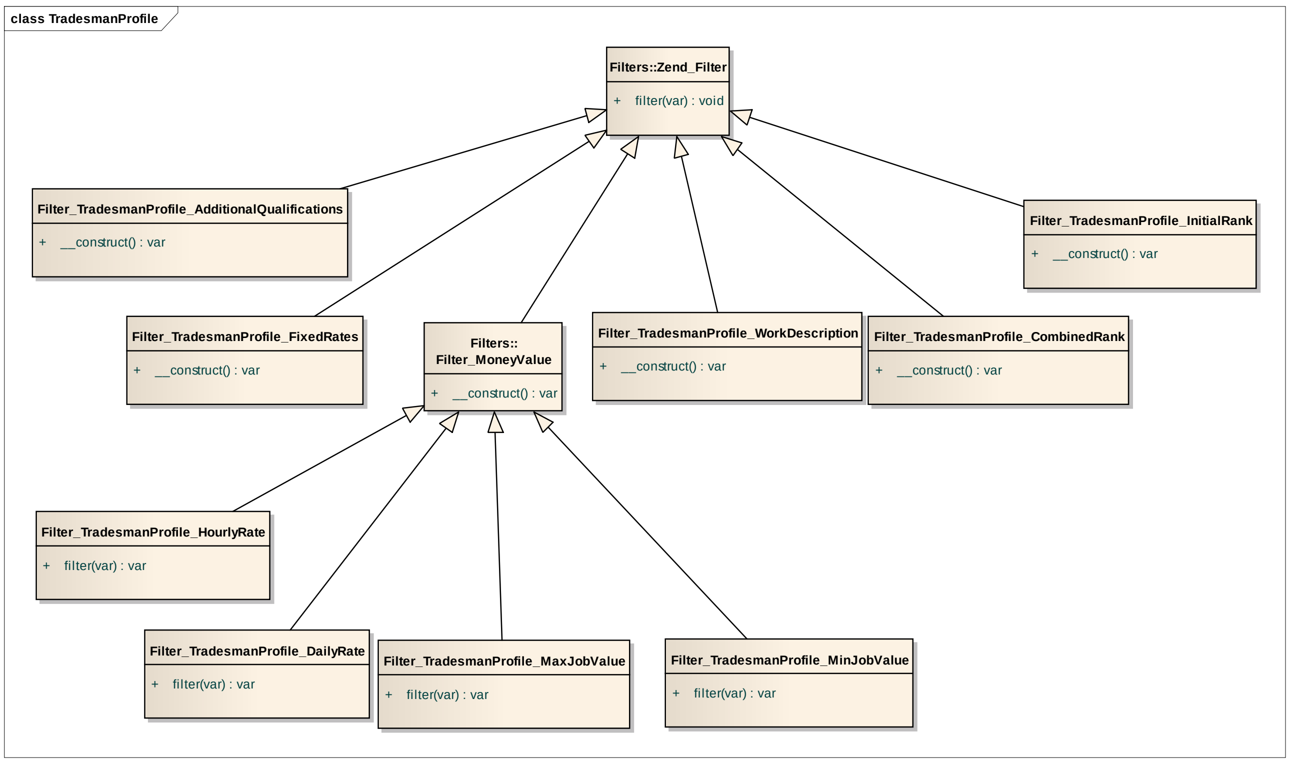
Planning Permission



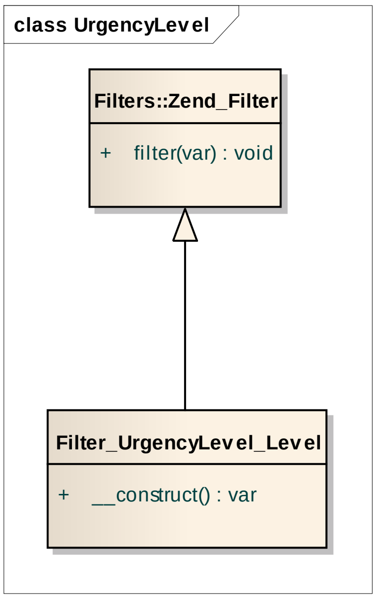
Qualification



Tradesman Profile


class TradesmanProfile
Filters::Zend_Filter
+ filter(var) : void
Filter_TradesmanProfile_AdditionalQualifications
+ __construct() : var
Filter_TradesmanProfile_InitialRank
+ __construct() : var
Filter_TradesmanProfile_FixedRates
+ __construct() : var
Filters::
Filter_MoneyValue
+ __construct() : var
Filter_TradesmanProfile_WorkDescription
+ __construct() : var
Filter_TradesmanProfile_CombinedRank
+ __construct() : var
Filter_TradesmanProfile_HourlyRate
+ filter(var) : var
Filter_TradesmanProfile_DailyRate
+ filter(var) : var
Filter_TradesmanProfile_MaxJobValue
+ filter(var) : var
Filter_TradesmanProfile_MinJobValue
+ filter(var) : var

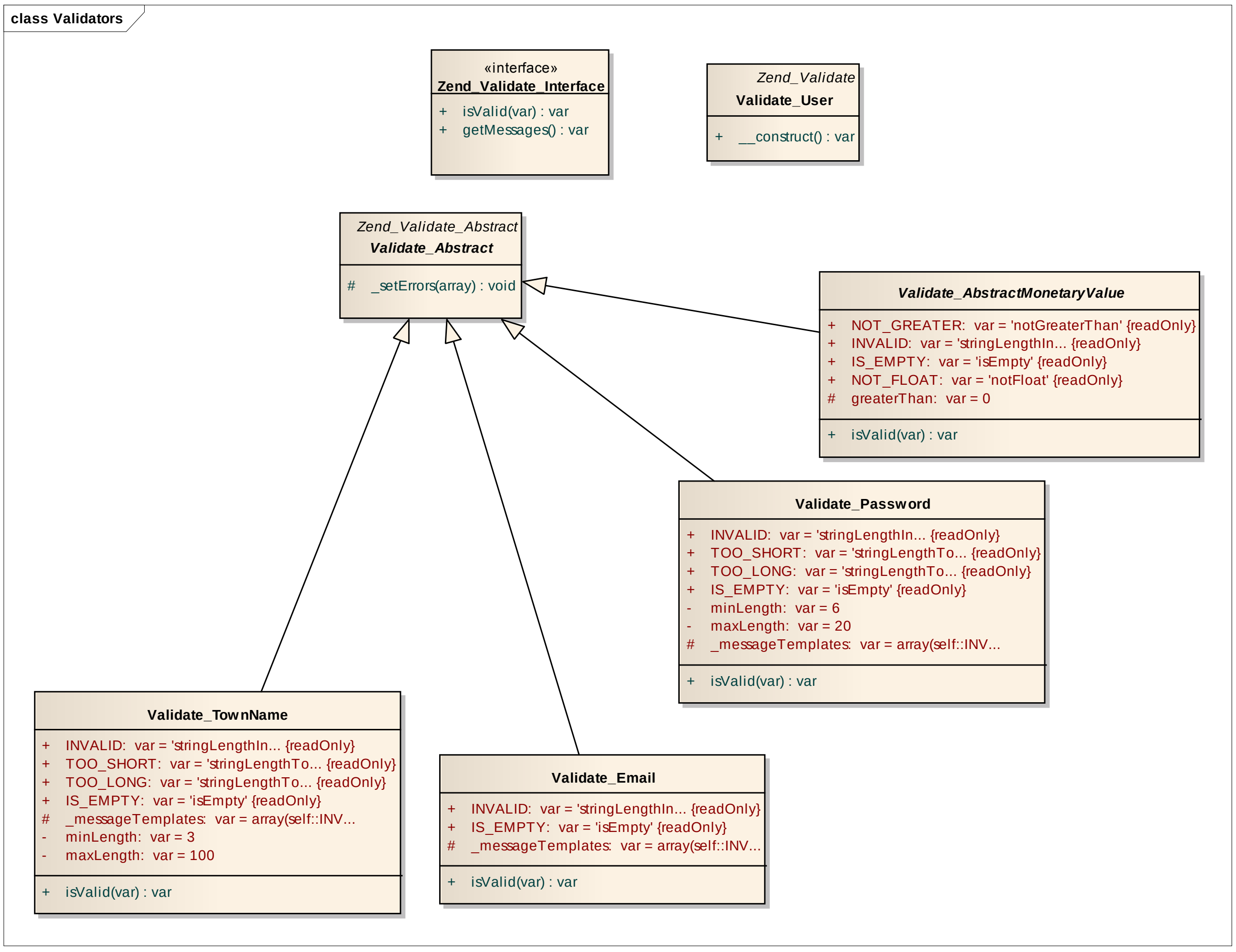
Urgency Level



Validators

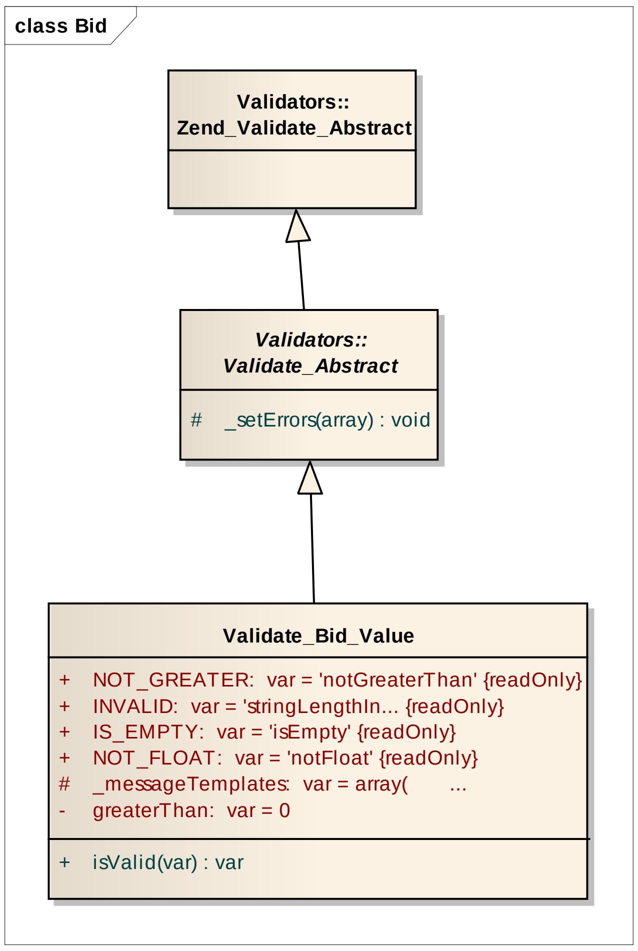
Validators ensure that provided data that comes from the user is valid before it gets persisted to the database. Validators work in conjunction with filters to achieve the desired result.

Generic

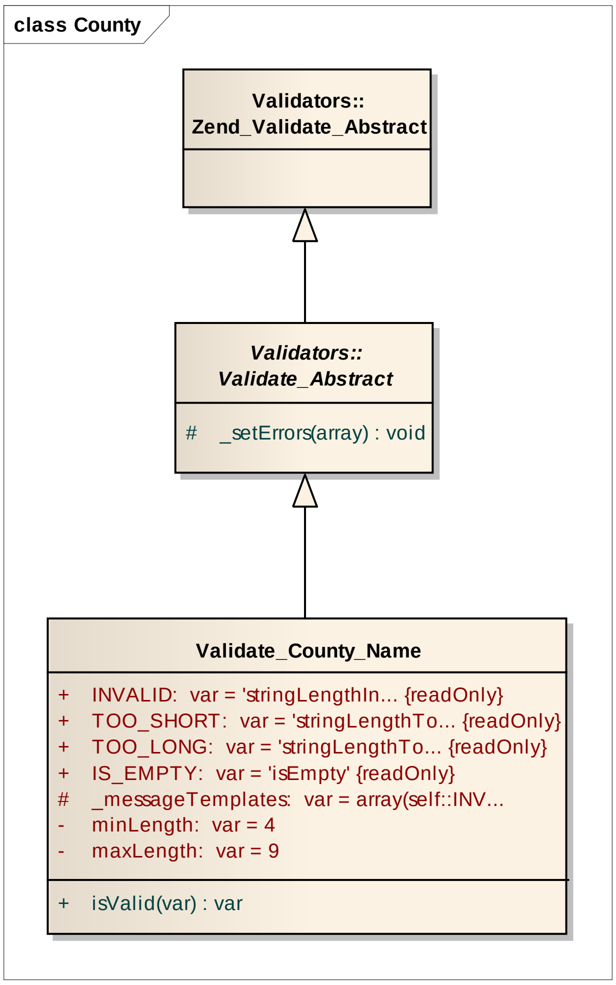


class Validators
«interface»
Zend_Validate_Interface
+ isValid(var) : var
+ getMessages() : var
Zend_Validate
Validate_User
+ __construct() : var
Zend_Validate_Abstract
Validate_Abstract
# _setErrors(array) : void
Validate_AbstractMonetaryValue
+ NOT_GREATER: var = 'notGreaterThan' {readOnly}
+ INVALID: var = 'stringLengthIn... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
+ NOT_FLOAT: var = 'notFloat' {readOnly}
# greaterThan: var = 0
+ isValid(var) : var
Validate_Password
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
- minLength: var = 6
- maxLength: var = 20
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validate_TownName
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 3
- maxLength: var = 100
+ isValid(var) : var
Validate_Email
+ INVALID: var = 'stringLengthIn... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var

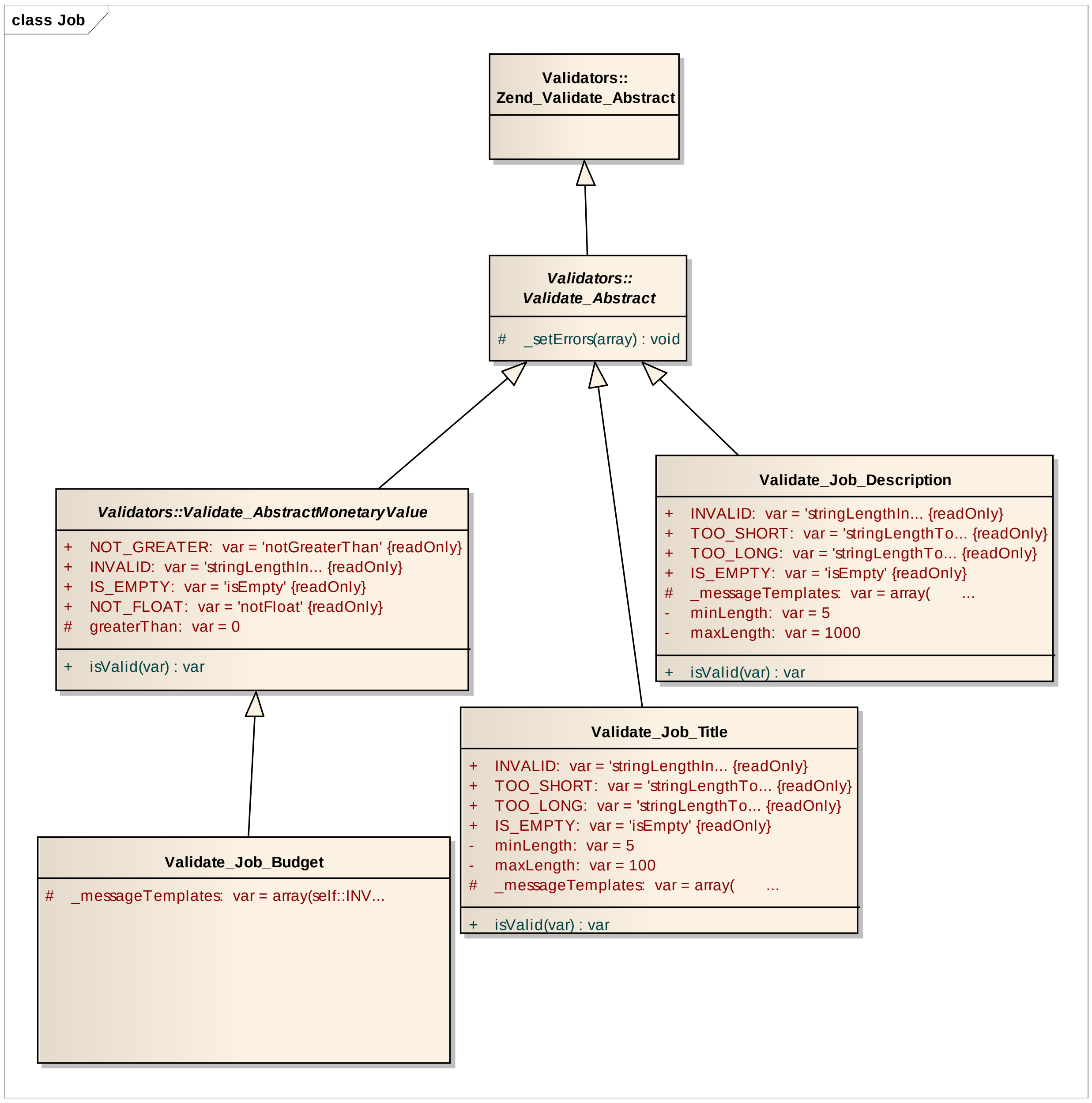
Bid


class Bid
Validators::
Zend_Validate_Abstract
Validators::
Validate_Abstract
# _setErrors(array) : void
Validate_Bid_Value
+ NOT_GREATER: var = 'notGreaterThan' {readOnly}
+ INVALID: var = 'stringLengthIn... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
+ NOT_FLOAT: var = 'notFloat' {readOnly}
# _messageTemplates: var = array(...
- greaterThan: var = 0
+ isValid(var) : var

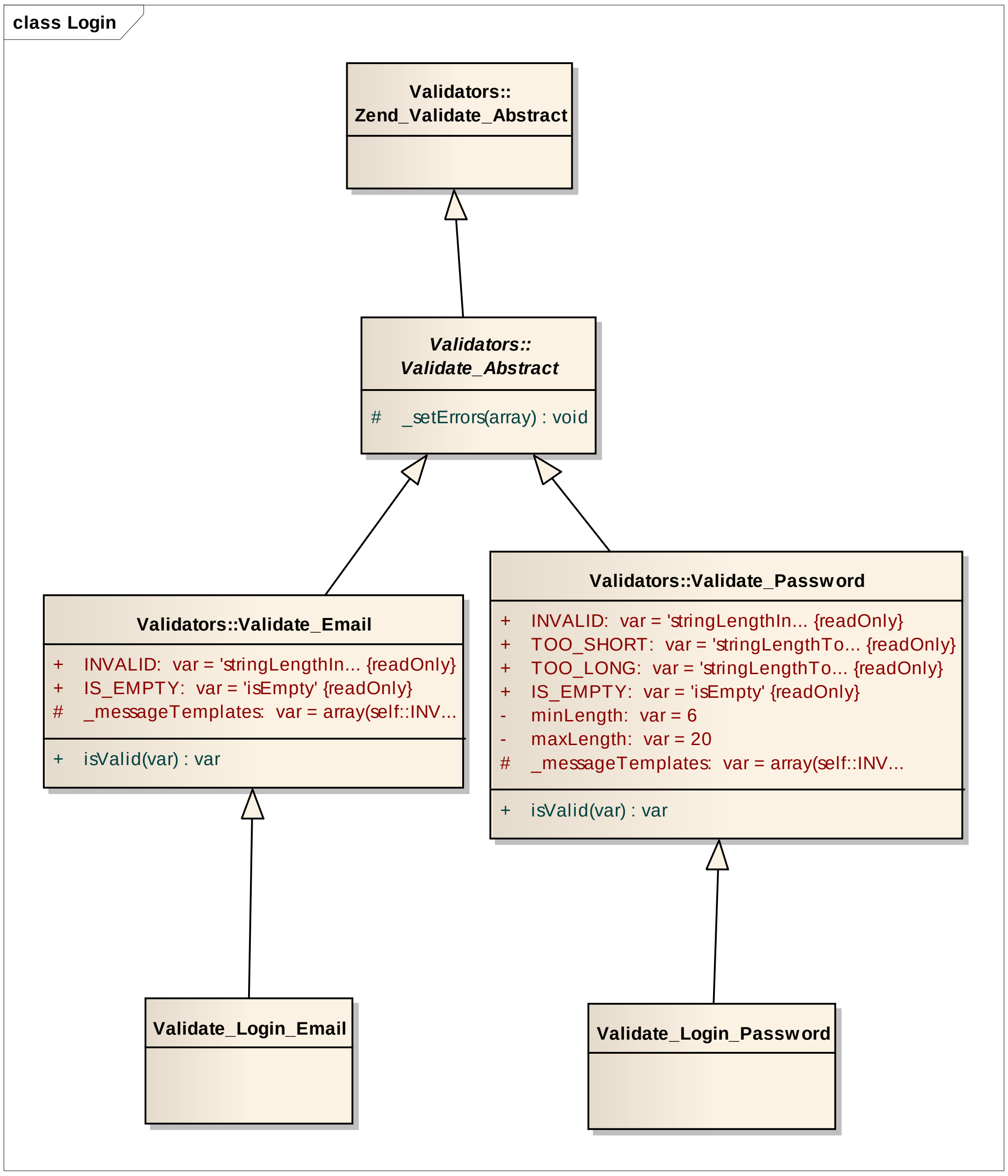
County


class County
Validators::
Zend_Validate_Abstract
Validators::
Validate_Abstract
# _setErrors(array) : void
Validate_County_Name
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 4
- maxLength: var = 9
+ isValid(var) : var

Job

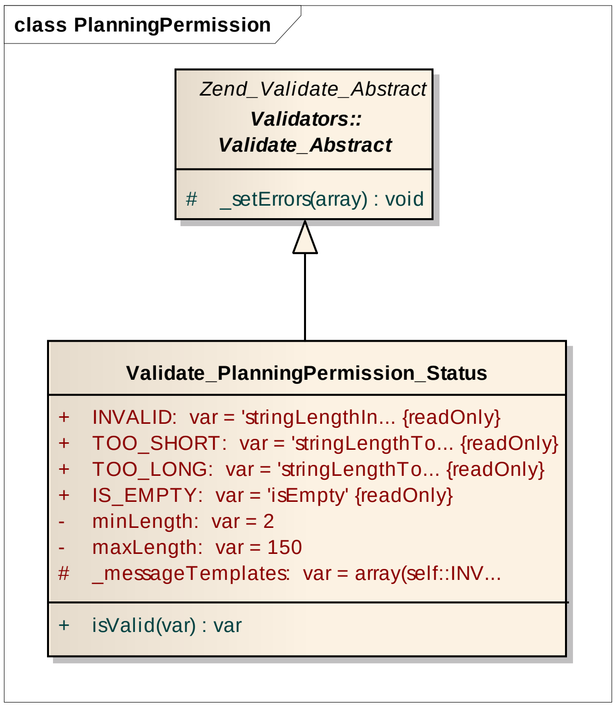

class Job
Validators::
Zend_Validate_Abstract
Validators::
Validate_Abstract
# _setErrors(array) : void
Validators::Validate_AbstractMonetaryValue
+ NOT_GREATER: var = 'notGreaterThan' {readOnly}
+ INVALID: var = 'stringLengthIn... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
+ NOT_FLOAT: var = 'notFloat' {readOnly}
# greaterThan: var = 0
+ isValid(var) : var
Validate_Job_Description
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(...
- minLength: var = 5
- maxLength: var = 1000
+ isValid(var) : var
Validate_Job_Title
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
- minLength: var = 5
- maxLength: var = 100
# _messageTemplates: var = array(...
+ isValid(var) : var
Validate_Job_Budget
# _messageTemplates: var = array(self::INV...

Login



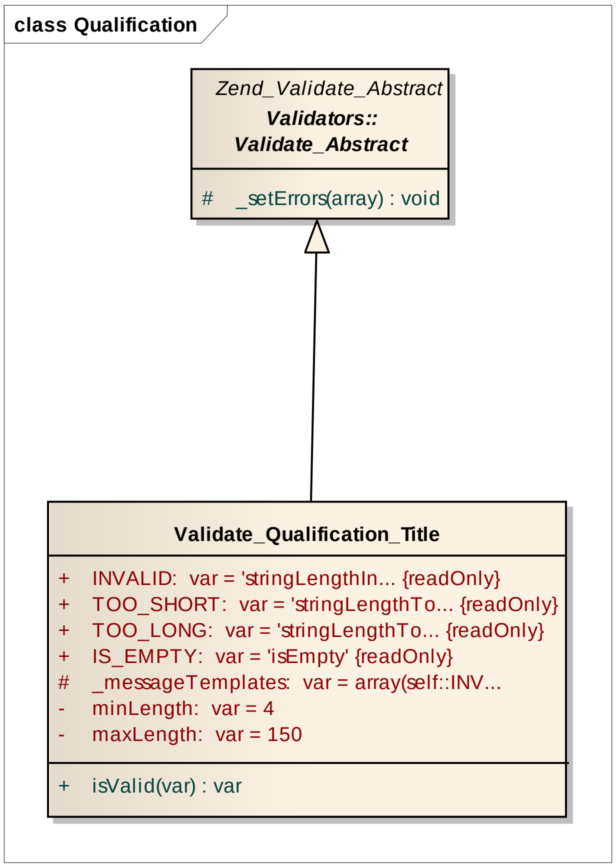
class Login
Validators:: Zend_Validate_Abstract
Validators:: Validate_Abstract
# _setErrors(array) : void
Validators::Validate_Email
+ INVALID: var = 'stringLengthIn... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validators::Validate_Password
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
- minLength: var = 6
- maxLength: var = 20
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validate_Login_Email
Validate_Login_Password

Planning Permission



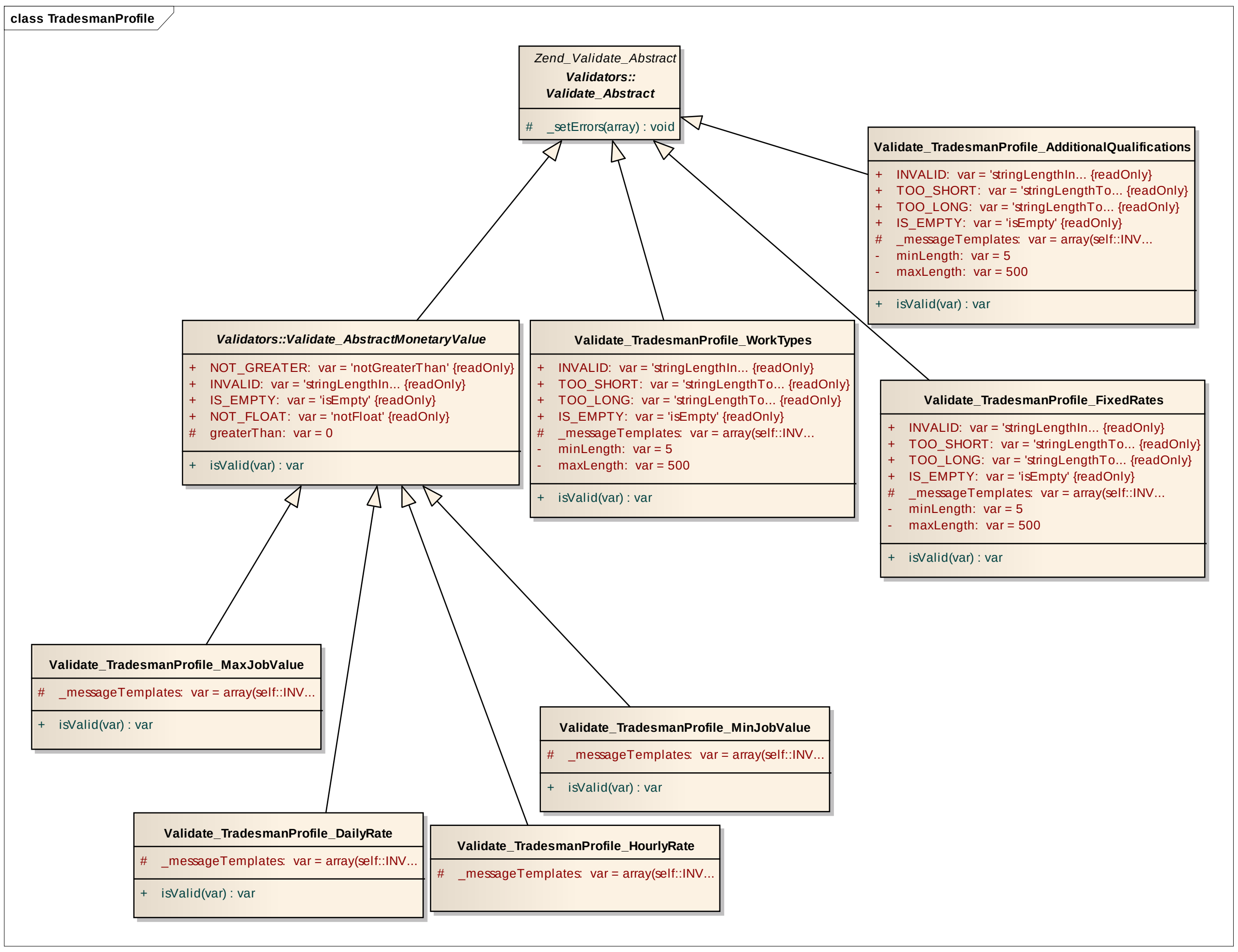
class PlanningPermission
Zend_Validate_Abstract
Validators::
Validate_Abstract
# _setErrors(array) : void
Validate_PlanningPermission_Status
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
- minLength: var = 2
- maxLength: var = 150
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var

Qualification

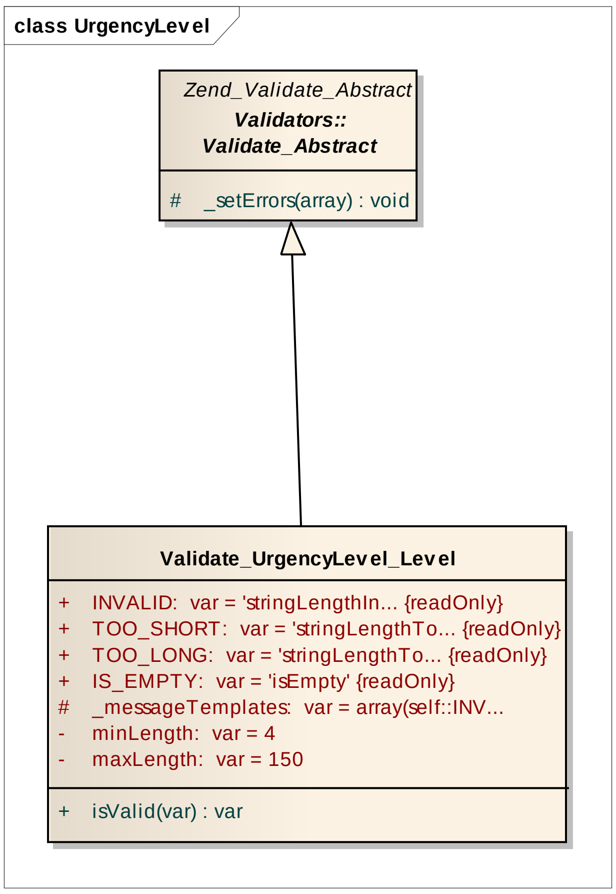


class Qualification
Zend_Validate_Abstract
Validators::
Validate_Abstract
# _setErrors(array) : void
Validate_Qualification_Title
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 4
- maxLength: var = 150
+ isValid(var) : var

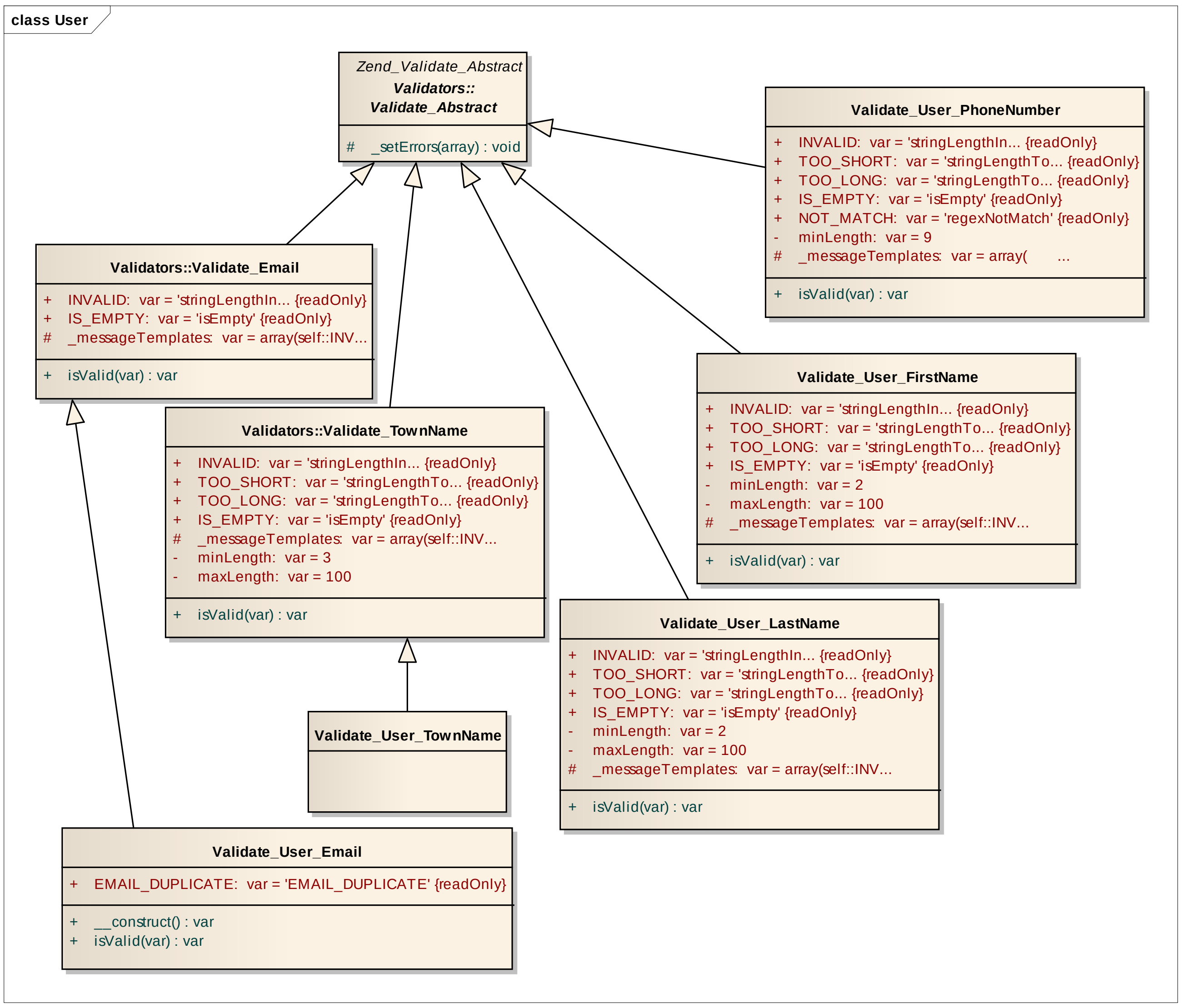
Tradesman Profile


class TradesmanProfile
Zend_Validate_Abstract
Validators::Validate_Abstract
# _setErrors(array) : void
Validate_TradesmanProfile_AdditionalQualifications
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 5
- maxLength: var = 500
+ isValid(var) : var
Validators::Validate_AbstractMonetaryValue
+ NOT_GREATER: var = 'notGreaterThan' {readOnly}
+ INVALID: var = 'stringLengthIn... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
+ NOT_FLOAT: var = 'notFloat' {readOnly}
# greaterThan: var = 0
+ isValid(var) : var
Validate_TradesmanProfile_WorkTypes
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 5
- maxLength: var = 500
+ isValid(var) : var
Validate_TradesmanProfile_FixedRates
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 5
- maxLength: var = 500
+ isValid(var) : var
Validate_TradesmanProfile_MaxJobValue
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validate_TradesmanProfile_MinJobValue
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validate_TradesmanProfile_DailyRate
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validate_TradesmanProfile_HourlyRate
# _messageTemplates: var = array(self::INV...

Urgency Level


class UrgencyLevel
Zend_Validate_Abstract
Validators::
Validate_Abstract
# _setErrors(array) : void
Validate_UrgencyLevel_Level
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 4
- maxLength: var = 150
+ isValid(var) : var

User

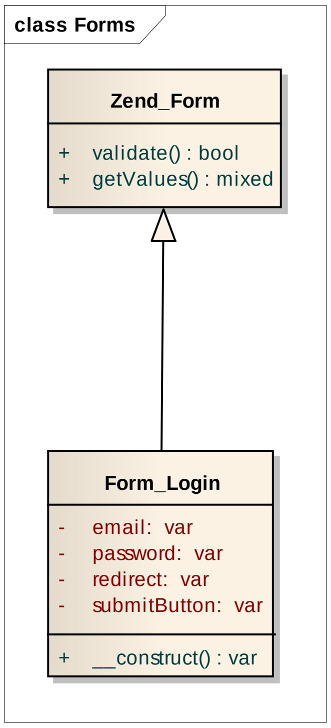


class User
Zend_Validate_Abstract
Validators::
Validate_Abstract
# _setErrors(array) : void
Validate_User_PhoneNumber
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
+ NOT_MATCH: var = 'regexNotMatch' {readOnly}
- minLength: var = 9
# _messageTemplates: var = array(...
+ isValid(var) : var
Validators::Validate_Email
+ INVALID: var = 'stringLengthIn... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validate_User_FirstName
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
- minLength: var = 2
- maxLength: var = 100
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validators::Validate_TownName
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
# _messageTemplates: var = array(self::INV...
- minLength: var = 3
- maxLength: var = 100
+ isValid(var) : var
Validate_User_LastName
+ INVALID: var = 'stringLengthIn... {readOnly}
+ TOO_SHORT: var = 'stringLengthTo... {readOnly}
+ TOO_LONG: var = 'stringLengthTo... {readOnly}
+ IS_EMPTY: var = 'isEmpty' {readOnly}
- minLength: var = 2
- maxLength: var = 100
# _messageTemplates: var = array(self::INV...
+ isValid(var) : var
Validate_User_TownName
Validate_User_Email
+ EMAIL_DUPLICATE: var = 'EMAIL_DUPLICATE' {readOnly}
+ __construct() : var
+ isValid(var) : var

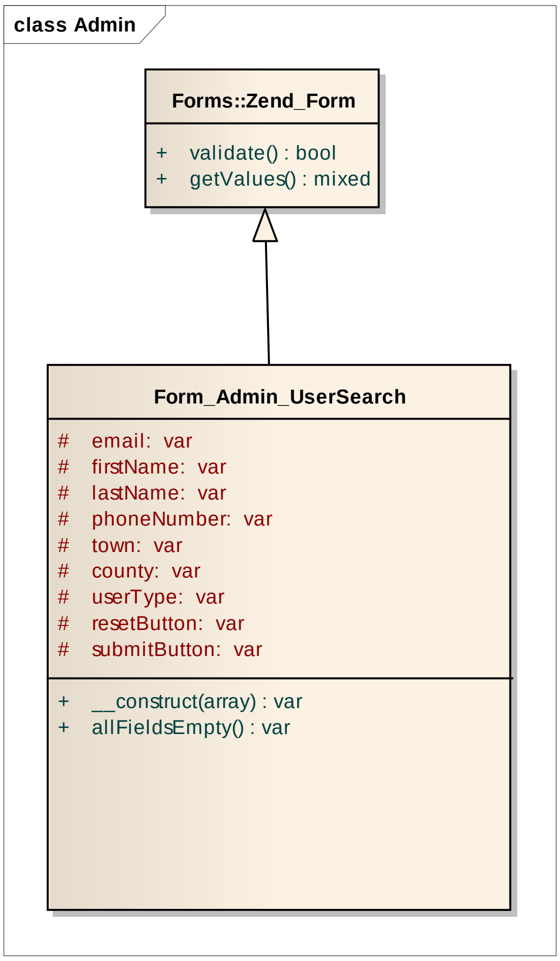
Forms

A form describes a HTML Form in PHP context – written in php and then rendered to the screen using decorators. A form encapsulates in itself a number of html fields (such as text, checkboxes, buttons, etc.) and a set of filters and validators for each of the fields (provided above). When the form is submitted, the data that was provided by the user can be fetched from the form in a state that it is already filtered and validated. Hence, ready to be mapped to an entity instance and persisted.

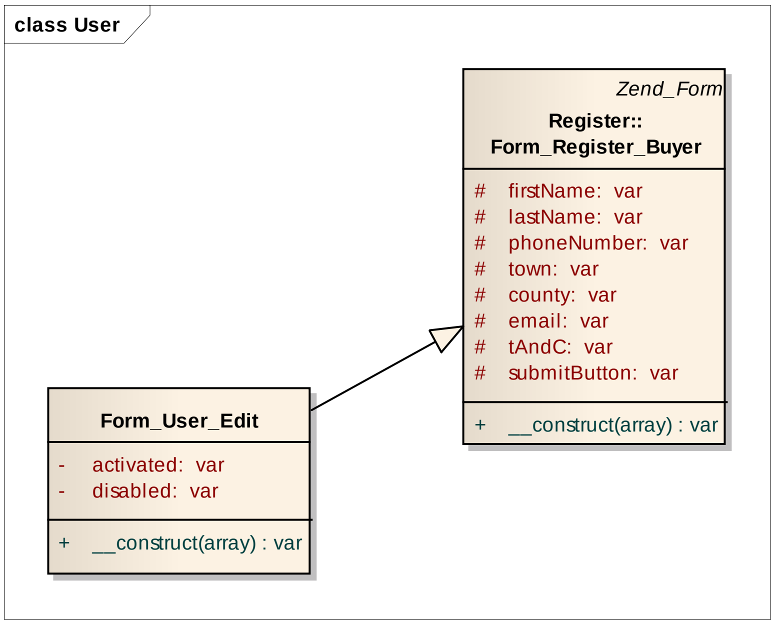
Login



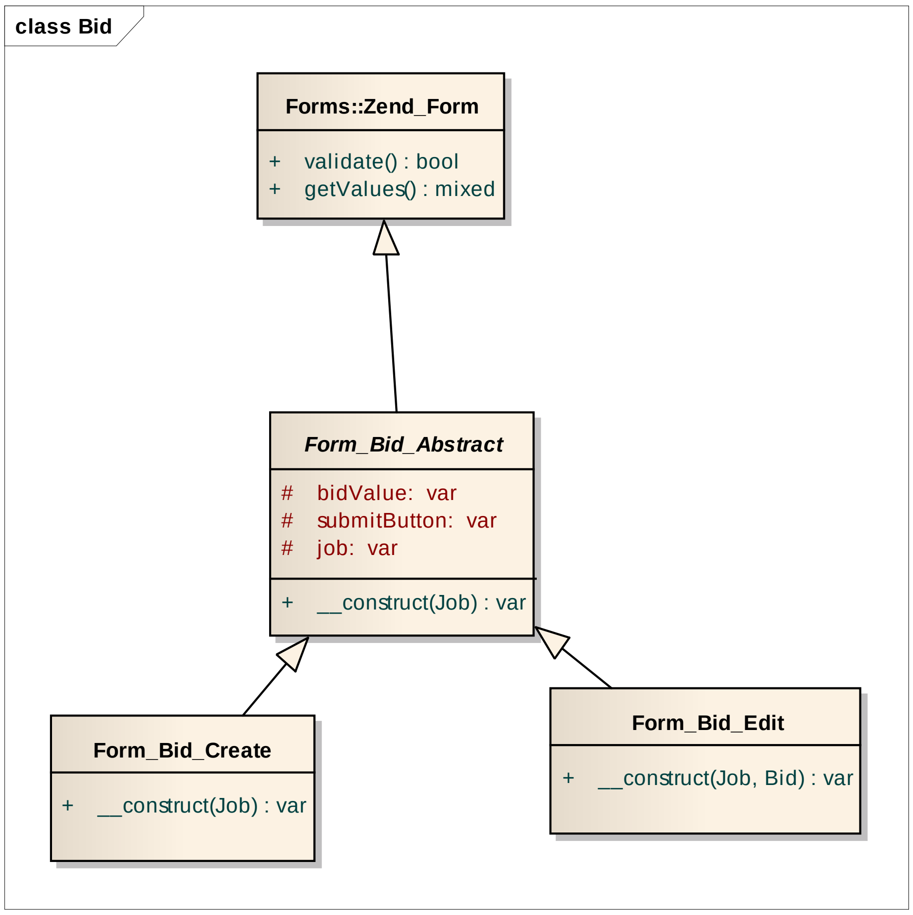
Admin – User Search



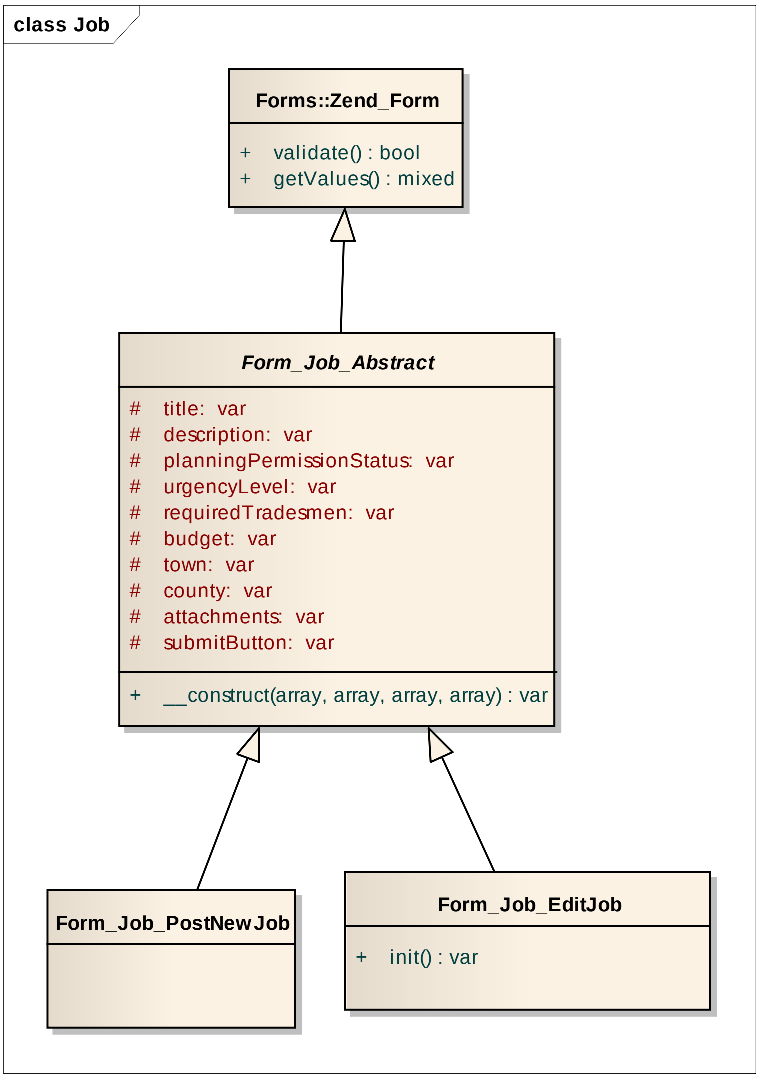
Admin – Edit User



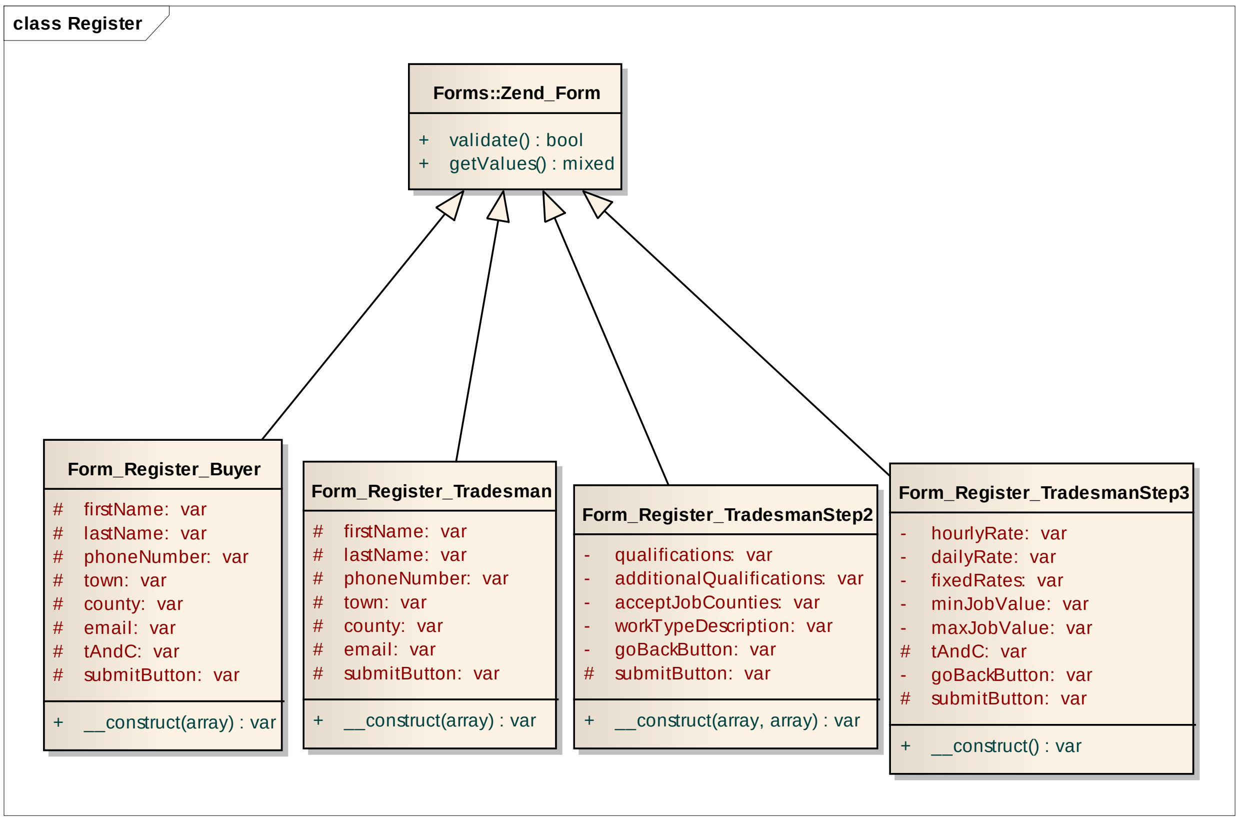
Bid


class Bid
Forms::Zend_Form
+ validate() : bool
+ getValues() : mixed
Form_Bid_Abstract
# bidValue: var
# submitButton: var
# job: var
+ __construct(Job) : var
Form_Bid_Create
+ __construct(Job) : var
Form_Bid_Edit
+ __construct(Job, Bid) : var

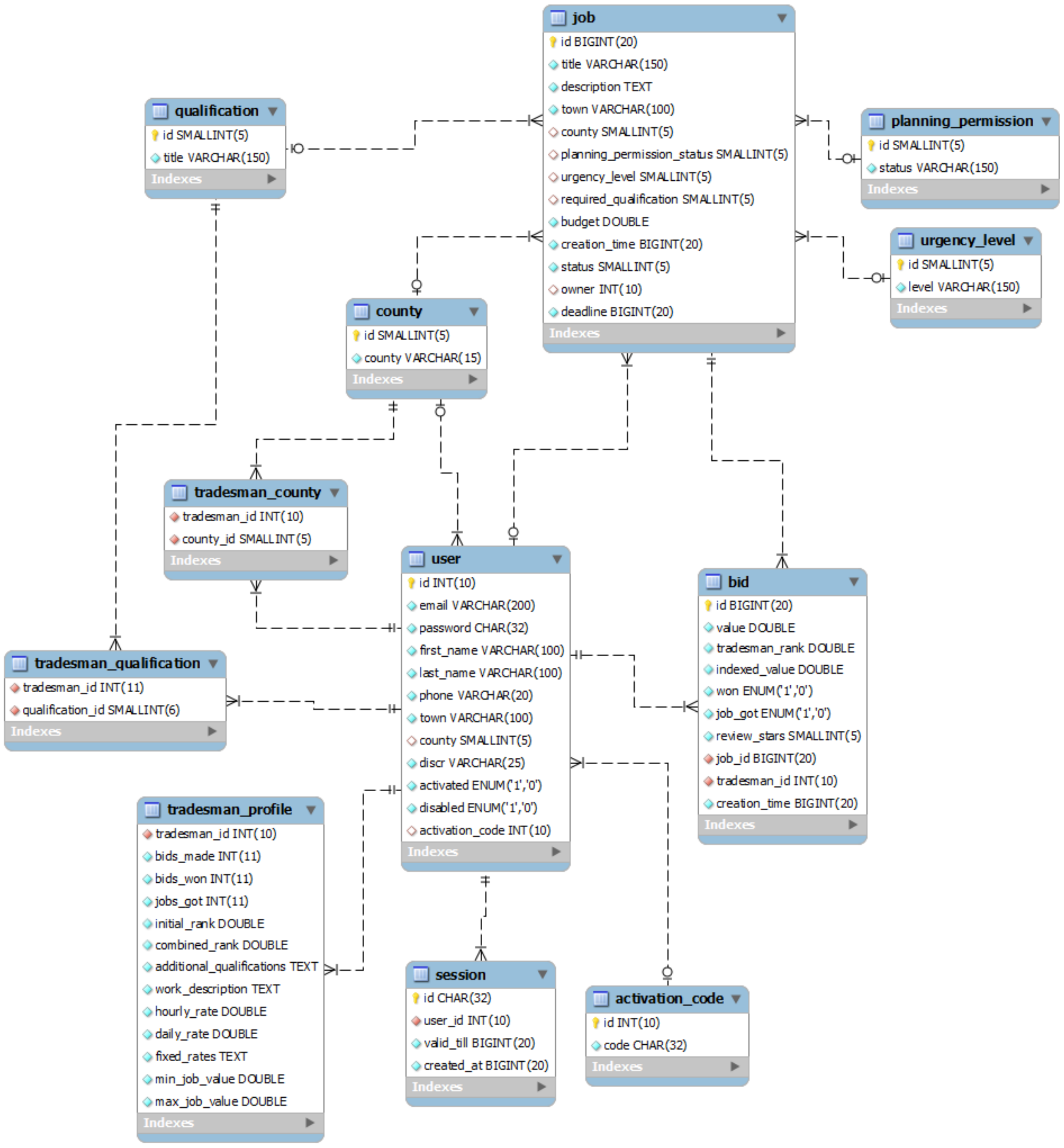
Job


class Job
Forms::Zend_Form
+ validate() : bool
+ getValues() : mixed
Form_Job_Abstract
# title: var
# description: var
# planningPermissionStatus: var
# urgencyLevel: var
# requiredTradesmen: var
# budget: var
# town: var
# county: var
# attachments: var
# submitButton: var
+ __construct(array, array, array, array) : var
Form_Job_PostNewJob
Form_Job_EditJob
+ init() : var

User Registration



Database Diagram



Testing

Unit testing using PHPUnit library will be performed for each Service Layer class and each method inside of it to ensure that for each of the uses cases the service layer produces correct results.

Tests will cover “happy path” and “rainy day scenarios” [1] for each use case implementation and compare expected results with the actual output. Such simple tests will be able to provide that for all identified scenarios the system behaves as expected.

[1] Iconix process terminology

Appendix

API Specification

The following is a specification that describes how to community to the system over HTTP protocol is a secure manner in order to consume the APIs that the system provides:

- Finalise a job and obtain the list of winners
- Update a job and specify the tradesman who was awarded the opportunity to physically carry out the job (to award them extra points for this win).

Finalising a Job

In order to obtain the job winners, a consumer can make a simple HTTP call to the following URL:

- **Url: /**api/index/finalise-job?**email**=x&**password**=y
  where email and password are login details of an administrator

With the given XML data:

```
<?xml version="1.0" encoding="UTF-8"?>
<job>
      <id>JobID:integer</id>
      <title>JobTitle:string</title>
      <description>JobDescription:text</description>
      <required_tradesmen>
            <id>QualificationID:integer</id>
            <title>QualificationTitle:string</title>
      </required_tradesmen>
      <planning_permission_status>
            <id>PlanningPermissionID:integer</</id>
            <title>PlanningPermissionTitle</title>
      </planning_permission_status>
      <urgency_level>
            <id>UrgencyLevelID:integer</id>
            <title>UrgencyLevelTitle</title>
      </urgency_level>
      <town>TownName:string</town>
      <county>CountyID:integer</county>
      <budget>Budget:double</budget>
      <owner>
            <id>UserID:integer</id>
      </owner>
      <bids>
            <bid>
                  <id>BidID:integer</id>
                  <value>BidValue:integer</value>
                  <tradesman>
                        <id>TradesmanID:integer</id>
                  </tradesman>
            </bid>
      </bids>
</job>
```

Between the *bids* tags, the consumer can specify any number of bids. All identifiers must be unique for all ID fields.

The system, in turn, will reply either with the list of bids or an error message.

```
<?xml version="1.0" encoding="UTF-8"?>
<reply>
      <outcome>{0|1}</outcome>
      <message>Message:string</message>
      <bids>
            <bid>
                  <id>23</id>
                  <value>16</value>
                  <index>1.016</index>
                  <indexedValue>16.256</indexedValue>

                  <value>16</value>
                  <won>1</won>
                  <jobGot></jobGot>
                  <reviewStars>0</reviewStars>
                  <tradesman>55</tradesman>
                  <creationTime>1297648847</creationTime>

            </bid>

            <bid>
                  <id>26</id>
                  <value>6</value>
                  <index>1.038</index>
                  <indexedValue>6.228</indexedValue>
                  <value>6</value>

                  <won>1</won>
                  <jobGot></jobGot>
                  <reviewStars>0</reviewStars>
                  <tradesman>58</tradesman>
                  <creationTime>1297648847</creationTime>
            </bid>

      </bids>
</reply>
```

The whole message is wrapped by the *reply* tag where the enclosed fields are:

- Outcome – {0|1}. 0 – an error occurred (check message tag). 1 – operation succeeded, see the bids.
- Message – an possible error message that describes what went wrong during algorithm execution
- Bids – the list of bids. All bids are sorted the assumed fitness of the tradesman for the job. So, the bid on top belongs to the tradesman who should be able to do the best job. The bid at the very bottom belongs to the tradesman who is least fit for the job.

Setting a winner

After the job was carried out, the API consumer will be able to specify the job winner and the review they obtained from customer.

```
<?xml version="1.0" encoding="UTF-8"?>
      <job>
          <id>JobID:integer</id>
          <bid>BidID:integer</bid>
          <review>ReviewStars{0-10}:integer</review>
   </job>
```

- ID – Job ID for which to set the winning bid
- BidID – the ID of the winning bid (the bid that got the job)
- ReviewStars – optional review stars. The rating that was given to the tradesman by the customer.

The system will reply in the template as given above:

- Outcome {1|0} – where 1 indicates a successful outcome of the operation and 0 indicates a failed operation
- Message – an optional error message that will be given if the outcome is 0.

Notification & Error Messages

Abbreviation	Text
ALREADY_LOGGED_IN	You are currently logged in. If you wish to create another account, please log out first.
SESSION_TERMINIATED	Your session has been successfully terminated.
SESSION_TIMEDOUT_REGISTRATION	Your session has timed out. Please restart the registration process.
LOGIN_FAILED	Incorrect Email or Password. Please try again.
ACCOUNT_NOT_ACTIVATED	Your account needs to be activated first.
ACCOUNT_DISABLED	Your account is disabled. Please contact the support to resolve this issue.
PRIV_ERROR_POST_JOB	You cannot post new jobs. Please register as a buyer to be able to post jobs.
NO_PRIV_EDIT_JOB	You do not have privileges to edit this job.
ACCT_REQ_POST_JOB	Please login or register to post a new job
NO_ACTIVE_JOBS	Currently, there are no active jobs. Please check back later.
REQ_JOB_NOT_FOUND	The requested job was not found.
NO_BIDS	There are no bids placed for this job
DELETE_CONF_JOB	Are you sure to want to delete [Job Title]?
BID_JOB_OVER	This job is finished. No more bids are accepted
BID_ALREADY_EXISTS	You already have a bid for this job. Please edit it instead.
ACCT_REQ_TO_PLACE_BIDS	Please log in as a tradesman or register a new account to place a bid.
NO_PRIV_EDIT_BID	You cannot edit this bid due to insufficient privileges.
NO_PRIV_DELETE_BID	You cannot delete this bid due to insufficient privileges.
NO_PRIV_DELETE_ACCOUNT	You cannot delete this account due to insufficient privileges.
NO_PRIV_DISABLE_ACCOUNT	You cannot disable this account due to insufficient privileges.
NO_PRIV_REMAIL_ACCOUNT	You cannot resend validation emails for this account due to insufficient privileges.
NO_PRIV_ACTIVATE_ACCOUNT	You cannot activate this account due to insufficient privileges.
REQ_BID_NOT_FOUND	The requested bid was not found
ACCOUNT_NOT_FOUND	The specified account was not found. Please try again.
ACCOUNT_ALREADY_ACTIVATED	The account is already activated
ACCT_REQ_EDIT_JOB	Please login to edit this job
EDIT_JOB_OVER	The job is finished and cannot be edited
NO_PRIV_DELETE_JOB	You do not have privileges to delete this job
NO_PRIV_EDIT_ACCOUNT	You do not have privileges to edit this account
NO_PRIV_ENABLE_ACCOUNT	You do not have privileges to enable this account
NO_PRIV_VIEW_ACCOUNT	You do not have privileges to view accounts