DAFTAR PUSTAKA

[1] Ilham, Kurnia, : Prinsip-prinsip otentikasi, FASILKOM; Universitas Indonesia 2006

[2] Carliner, Saul; Shank, Patti, The E-learning Handbook, San Fransisco: J hon Willey and Sons 2008

[3] Rosen, Anita, *E-Learning* 2.0 : *proven practices and emerging technologies to achieve results*, New York: AMACOM 2009

[4] Kozlowski ,Timo. E-Learning 1.0 and E-learning 2.0, *Paper for the International E-Learning Conference at the Rajabhat Suan Dusit University*.

[5] Chao, Lee, *Utilizing open source tools for on-line teaching and learning*: *appliying linux technologies*, Hersey PA: Information Science Reference 2009

[6] Ramadhana, Yovira., Sistem Konferensi Video Berbasis Web dengan Menggunakan Aplikasi *Open Source*, *Freeware* dan *Shareware*, Jakarta: Universitas Trisakti 2008

[7] Sri, Dharwiyanti, Pengantar *Unified Modelling Languange* (UML), 2003

[8] Rudy, Richie, Odi, Gunadi.,: Integrasi Aplikasi Menggunakan Singel Sign On Berbasiskan *Leightweight Directory Access Protocol* (LDAP) Dalam Portal, Universitas Bina Nusantara, 2009

[9] Amak, Yunus, Implementasi Sistem Otentikasi Pada Pengguna Jaringan *Hotspot* di Universitas Kanjuruhan Malang Guna Meningkatkan Keamanan Jaringan Komputer.

[10] Yesi, Novaria, Kunang., Iman, Zuhri, Yadi.,: Pengembangan Sistem Autentikasi *Hotspot* Akademis Terpusat Berbasis Teknologi *Web Service*, 2012

```
// ########################################################################
//  AUTHENTICATION
// ########################################################################
        /**
         * @addtogroup publicAuth
         * @{
         */
        /**
         * Set the times authentication will be cached before really accessing the CAS server in
gateway mode:
         * - -1: check only once, and then never again (until you pree login)
         * - 0: always check
         * - n: check every "n" time
         *
         * @param $n an integer.
         */
        function setCacheTimesForAuthRecheck($n) {
                global $PHPCAS_CLIENT;
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
                }
                if (gettype($n) != 'integer') {
                        phpCAS :: error('type mismatched for parameter $header (should be
`string\')');
                }
                $PHPCAS_CLIENT->setCacheTimesForAuthRecheck($n);
        }
        /**
         * This method is called to check if the user is authenticated (use the gateway feature).
         * @return TRUE when the user is authenticated; otherwise FALSE.
         */
        function checkAuthentication() {
                global $PHPCAS_CLIENT, $PHPCAS_AUTH_CHECK_CALL;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
                }
                $auth = $PHPCAS_CLIENT->checkAuthentication();
                // store where the authentication has been checked and the result
                $dbg = phpCAS :: backtrace();
                $PHPCAS_AUTH_CHECK_CALL = array (
                        'done' => TRUE,
                        'file' => $dbg[0]['file'],
                        'line' => $dbg[0]['line'],
                        'method' => __CLASS__ . '::' . __FUNCTION__,
                        'result' => $auth
                );
                phpCAS :: traceEnd($auth);
                return $auth;
        }
        /**
         * This method is called to force authentication if the user was not already
         * authenticated. If the user is not authenticated, halt by redirecting to
         * the CAS server.
         */
        function forceAuthentication() {
                global $PHPCAS_CLIENT, $PHPCAS_AUTH_CHECK_CALL;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
```

```
                        phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
                }
                $auth = $PHPCAS_CLIENT->forceAuthentication();
                // store where the authentication has been checked and the result
                $dbg = phpCAS :: backtrace();
                $PHPCAS_AUTH_CHECK_CALL = array (
                        'done' => TRUE,
                        'file' => $dbg[0]['file'],
                        'line' => $dbg[0]['line'],
                        'method' => __CLASS__ . '::' . __FUNCTION__,
                        'result' => $auth
                );
                if (!$auth) {
                        phpCAS :: trace('user is not authenticated, redirecting to the CAS
server');
                        $PHPCAS_CLIENT->forceAuthentication();
                } else {
                        phpCAS :: trace('no need to authenticate (user `' . phpCAS :: getUser() .
'\' is already authenticated)');
                }
                phpCAS :: traceEnd();
                return $auth;
        }
        /**
         * This method is called to renew the authentication.
         **/
        function renewAuthentication() {
                global $PHPCAS_CLIENT, $PHPCAS_AUTH_CHECK_CALL;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should not be called before' . __CLASS__ .
'::client() or ' . __CLASS__ . '::proxy()');
                }
                // store where the authentication has been checked and the result
                $dbg = phpCAS :: backtrace();
                $PHPCAS_AUTH_CHECK_CALL = array (
                        'done' => TRUE,
                        'file' => $dbg[0]['file'],
                        'line' => $dbg[0]['line'],
                        'method' => __CLASS__ . '::' . __FUNCTION__,
                        'result' => $auth
                );

                $PHPCAS_CLIENT->renewAuthentication();
                phpCAS :: traceEnd();
        }
        /**
         * This method has been left from version 0.4.1 for compatibility reasons.
         */
        function authenticate() {
                phpCAS :: error('this method is deprecated. You should use ' . __CLASS__ .
'::forceAuthentication() instead');
        }
        /**
         * This method is called to check if the user is authenticated (previously or by
         * tickets given in the URL).
         *
         * @return TRUE when the user is authenticated.
         */
        function isAuthenticated() {
                global $PHPCAS_CLIENT, $PHPCAS_AUTH_CHECK_CALL;
```

```
                        phpCAS :: traceBegin();
                        if (!is_object($PHPCAS_CLIENT)) {
                                phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
                        }
                        // call the isAuthenticated method of the global $PHPCAS_CLIENT object
                        $auth = $PHPCAS_CLIENT->isAuthenticated();

                        // store where the authentication has been checked and the result
                        $dbg = phpCAS :: backtrace();
                        $PHPCAS_AUTH_CHECK_CALL = array (
                                'done' => TRUE,
                                'file' => $dbg[0]['file'],
                                'line' => $dbg[0]['line'],
                                'method' => __CLASS__ . '::' . __FUNCTION__,
                                'result' => $auth
                        );
                        phpCAS :: traceEnd($auth);
                        return $auth;
                }
                /**
                 * Checks whether authenticated based on $_SESSION. Useful to avoid
                 * server calls.
                 * @return true if authenticated, false otherwise.
                 * @since 0.4.22 by Brendan Arnold
                 */
                function isSessionAuthenticated() {
                        global $PHPCAS_CLIENT;
                        if (!is_object($PHPCAS_CLIENT)) {
                                phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
                        }
                        return ($PHPCAS_CLIENT->isSessionAuthenticated());
                }
                /**
                 * This method returns the CAS user's login name.
                 * @warning should not be called only after phpCAS::forceAuthentication()
                 * or phpCAS::checkAuthentication().
                 *
                 * @return the login name of the authenticated user
                 */
                function getUser() {
                        global $PHPCAS_CLIENT, $PHPCAS_AUTH_CHECK_CALL;
                        if (!is_object($PHPCAS_CLIENT)) {
                                phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
                        }
                        if (!$PHPCAS_AUTH_CHECK_CALL['done']) {
                                phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::forceAuthentication() or ' . __CLASS__ . '::isAuthenticated()');
                        }
                        if (!$PHPCAS_AUTH_CHECK_CALL['result']) {
                                phpCAS :: error('authentication was checked (by ' .
$PHPCAS_AUTH_CHECK_CALL['method'] . '() at ' . $PHPCAS_AUTH_CHECK_CALL['file'] . ':' .
$PHPCAS_AUTH_CHECK_CALL['line'] . ') but the method returned FALSE');
                        }
                        return $PHPCAS_CLIENT->getUser();
                }
                /**
                 * This method returns the CAS user's login name.
                 * @warning should not be called only after phpCAS::forceAuthentication()
                 * or phpCAS::checkAuthentication().
```

```
	 *
	 * @return the login name of the authenticated user
	 */
	function getAttributes() {
		global $PHPCAS_CLIENT, $PHPCAS_AUTH_CHECK_CALL;
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
		}
		if (!$PHPCAS_AUTH_CHECK_CALL['done']) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::forceAuthentication() or ' . __CLASS__ . '::isAuthenticated()');
		}
		if (!$PHPCAS_AUTH_CHECK_CALL['result']) {
			phpCAS :: error('authentication was checked (by ' .
$PHPCAS_AUTH_CHECK_CALL['method'] . '() at ' . $PHPCAS_AUTH_CHECK_CALL['file'] . ':' .
$PHPCAS_AUTH_CHECK_CALL['line'] . ') but the method returned FALSE');
		}
		return $PHPCAS_CLIENT->getAttributes();
	}
	/**
	 * Handle logout requests.
	 */
	function handleLogoutRequests($check_client = true, $allowed_clients = false) {
		global $PHPCAS_CLIENT;
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
		}
		return ($PHPCAS_CLIENT->handleLogoutRequests($check_client,
$allowed_clients));
	}
	/**
	 * This method returns the URL to be used to login.
	 * or phpCAS::isAuthenticated().
	 *
	 * @return the login name of the authenticated user
	 */
	function getServerLoginURL() {
		global $PHPCAS_CLIENT;
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
		}
		return $PHPCAS_CLIENT->getServerLoginURL();
	}
	/**
	 * Set the login URL of the CAS server.
	 * @param $url the login URL
	 * @since 0.4.21 by Wyman Chan
	 */
	function setServerLoginURL($url = '') {
		global $PHPCAS_CLIENT;
		phpCAS :: traceBegin();
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after
' . __CLASS__ . '::client()');
		}
		if (gettype($url) != 'string') {
			phpCAS :: error('type mismatched for parameter $url (should be
`string\')');
		}
```

```
        $PHPCAS_CLIENT->setServerLoginURL($url);
        phpCAS :: traceEnd();
}
/**
 * Set the serviceValidate URL of the CAS server.
 * Used only in CAS 1.0 validations
 * @param $url the serviceValidate URL
 * @since 1.1.0 by Joachim Fritschi
 */
function setServerServiceValidateURL($url = '') {
        global $PHPCAS_CLIENT;
        phpCAS :: traceBegin();
        if (!is_object($PHPCAS_CLIENT)) {
                phpCAS :: error('this method should only be called after
                                                        ' . __CLASS__ . '::client()');
        }
        if (gettype($url) != 'string') {
                phpCAS :: error('type mismatched for parameter $url (should be
                                                `string\')');
        }
        $PHPCAS_CLIENT->setServerServiceValidateURL($url);
        phpCAS :: traceEnd();
}
/**
 * Set the proxyValidate URL of the CAS server.
 * Used for all CAS 2.0 validations
 * @param $url the proxyValidate URL
 * @since 1.1.0 by Joachim Fritschi
 */
function setServerProxyValidateURL($url = '') {
        global $PHPCAS_CLIENT;
        phpCAS :: traceBegin();
        if (!is_object($PHPCAS_CLIENT)) {
                phpCAS :: error('this method should only be called after
                                                        ' . __CLASS__ . '::client()');
        }
        if (gettype($url) != 'string') {
                phpCAS :: error('type mismatched for parameter $url (should be
                                                `string\')');
        }
        $PHPCAS_CLIENT->setServerProxyValidateURL($url);
        phpCAS :: traceEnd();
}
/**
 * Set the samlValidate URL of the CAS server.
 * @param $url the samlValidate URL
 * @since 1.1.0 by Joachim Fritschi
 */
function setServerSamlValidateURL($url = '') {
        global $PHPCAS_CLIENT;
        phpCAS :: traceBegin();
        if (!is_object($PHPCAS_CLIENT)) {
                phpCAS :: error('this method should only be called after
                                                        ' . __CLASS__ . '::client()');
        }
        if (gettype($url) != 'string') {
                phpCAS :: error('type mismatched for parameter $url (should be
                                                `string\')');
        }
        $PHPCAS_CLIENT->setServerSamlValidateURL($url);
        phpCAS :: traceEnd();
}
```

```
	/**
	 * This method returns the URL to be used to login.
	 * or phpCAS::isAuthenticated().
	 *
	 * @return the login name of the authenticated user
	 */
	function getServerLogoutURL() {
		global $PHPCAS_CLIENT;
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should not be called before ' . __CLASS__
. '::client() or ' . __CLASS__ . '::proxy()');
		}
		return $PHPCAS_CLIENT->getServerLogoutURL();
	}
	/**
	 * Set the logout URL of the CAS server.
	 * @param $url the logout URL
	 * @since 0.4.21 by Wyman Chan
	 */
	function setServerLogoutURL($url = '') {
		global $PHPCAS_CLIENT;
		phpCAS :: traceBegin();
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after
' . __CLASS__ . '::client()');
		}
		if (gettype($url) != 'string') {
			phpCAS :: error('type mismatched for parameter $url (should be
`string\')');
		}
		$PHPCAS_CLIENT->setServerLogoutURL($url);
		phpCAS :: traceEnd();
	}
	/**
	 * This method is used to logout from CAS.
	 * @params $params an array that contains the optional url and service parameters that
will be passed to the CAS server
	 * @public
	 */
	function logout($params = "") {
		global $PHPCAS_CLIENT;
		phpCAS :: traceBegin();
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
		}
		$parsedParams = array ();
		if ($params != "") {
			if (is_string($params)) {
				phpCAS :: error('method `phpCAS::logout($url)\' is now
deprecated, use `phpCAS::logoutWithUrl($url)\' instead');
			}
			if (!is_array($params)) {
				phpCAS :: error('type mismatched for parameter $params
(should be `array\')');
			}
			foreach ($params as $key => $value) {
				if ($key != "service" && $key != "url") {
					phpCAS :: error('only `url\' and `service\' parameters
are allowed for method `phpCAS::logout($params)\'');
				}
				$parsedParams[$key] = $value;
```

```
                        }
                }
                $PHPCAS_CLIENT->logout($parsedParams);
                // never reached
                phpCAS :: traceEnd();
        }
        /**
         * This method is used to logout from CAS. Halts by redirecting to the CAS server.
         * @param $service a URL that will be transmitted to the CAS server
         */
        function logoutWithRedirectService($service) {
                global $PHPCAS_CLIENT;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
                }
                if (!is_string($service)) {
                        phpCAS :: error('type mismatched for parameter $service (should be
`string\')');
                }
                $PHPCAS_CLIENT->logout(array (
                        "service" => $service
                ));
                // never reached
                phpCAS :: traceEnd();
        }
        /**
         * This method is used to logout from CAS. Halts by redirecting to the CAS server.
         * @param $url a URL that will be transmitted to the CAS server
         */
        function logoutWithUrl($url) {
                global $PHPCAS_CLIENT;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
                }
                if (!is_string($url)) {
                        phpCAS :: error('type mismatched for parameter $url (should be
`string\')');
                }
                $PHPCAS_CLIENT->logout(array (
                        "url" => $url
                ));
                // never reached
                phpCAS :: traceEnd();
        }
        /**
         * This method is used to logout from CAS. Halts by redirecting to the CAS server.
         * @param $service a URL that will be transmitted to the CAS server
         * @param $url a URL that will be transmitted to the CAS server
         */
        function logoutWithRedirectServiceAndUrl($service, $url) {
                global $PHPCAS_CLIENT;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
                }
                if (!is_string($service)) {
```

```
                        phpCAS :: error('type mismatched for parameter $service (should be
`string\')');
                }
                if (!is_string($url)) {
                        phpCAS :: error('type mismatched for parameter $url (should be
`string\')');
                }
                $PHPCAS_CLIENT->logout(array (
                        "service" => $service,
                        "url" => $url
                ));
                // never reached
                phpCAS :: traceEnd();
        }
        /**
         * Set the fixed URL that will be used by the CAS server to transmit the PGT.
         * When this method is not called, a phpCAS script uses its own URL for the callback.
         *
         * @param $url the URL
         */
        function setFixedCallbackURL($url = '') {
                global $PHPCAS_CLIENT;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::proxy()');
                }
                if (!$PHPCAS_CLIENT->isProxy()) {
                        phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::proxy()');
                }
                if (gettype($url) != 'string') {
                        phpCAS :: error('type mismatched for parameter $url (should be
`string\')');
                }
                $PHPCAS_CLIENT->setCallbackURL($url);
                phpCAS :: traceEnd();
        }
        /**
         * Set the fixed URL that will be set as the CAS service parameter. When this
         * method is not called, a phpCAS script uses its own URL.
         *
         * @param $url the URL
         */
        function setFixedServiceURL($url) {
                global $PHPCAS_CLIENT;
                phpCAS :: traceBegin();
                if (!is_object($PHPCAS_CLIENT)) {
                        phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::proxy()');
                }
                if (gettype($url) != 'string') {
                        phpCAS :: error('type mismatched for parameter $url (should be
`string\')');
                }
                $PHPCAS_CLIENT->setURL($url);
                phpCAS :: traceEnd();
        }
        /**
         * Get the URL that is set as the CAS service parameter.
         */
        function getServiceURL() {
```

```
		global $PHPCAS_CLIENT;
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::proxy()');
		}
		return ($PHPCAS_CLIENT->getURL());
	}
	/**
	 * Retrieve a Proxy Ticket from the CAS server.
	 */
	function retrievePT($target_service, & $err_code, & $err_msg) {
		global $PHPCAS_CLIENT;
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::proxy()');
		}
		if (gettype($target_service) != 'string') {
			phpCAS :: error('type mismatched for parameter $target_service(should
be `string\')');
		}
		return ($PHPCAS_CLIENT->retrievePT($target_service, $err_code, $err_msg));
	}
	/**
	 * Set the certificate of the CAS server.
	 *
	 * @param $cert the PEM certificate
	 */
	function setCasServerCert($cert) {
		global $PHPCAS_CLIENT;
		phpCAS :: traceBegin();
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
		}
		if (gettype($cert) != 'string') {
			phpCAS :: error('type mismatched for parameter $cert (should be
`string\')');
		}
		$PHPCAS_CLIENT->setCasServerCert($cert);
		phpCAS :: traceEnd();
	}
	/**
	 * Set the certificate of the CAS server CA.
	 *
	 * @param $cert the CA certificate
	 */
	function setCasServerCACert($cert) {
		global $PHPCAS_CLIENT;
		phpCAS :: traceBegin();
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
		}
		if (gettype($cert) != 'string') {
			phpCAS :: error('type mismatched for parameter $cert (should be
`string\')');
		}
		$PHPCAS_CLIENT->setCasServerCACert($cert);
		phpCAS :: traceEnd();
	}
	/**
	 * Set no SSL validation for the CAS server.
```

```
	 */
	function setNoCasServerValidation() {
		global $PHPCAS_CLIENT;
		phpCAS :: traceBegin();
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
		}
		$PHPCAS_CLIENT->setNoCasServerValidation();
		phpCAS :: traceEnd();
	}
	/** @} */
	/**
	 * Change CURL options.
	 * CURL is used to connect through HTTPS to CAS server
	 * @param $key the option key
	 * @param $value the value to set
	 */
	function setExtraCurlOption($key, $value) {
		global $PHPCAS_CLIENT;
		phpCAS :: traceBegin();
		if (!is_object($PHPCAS_CLIENT)) {
			phpCAS :: error('this method should only be called after ' . __CLASS__ .
'::client() or' . __CLASS__ . '::proxy()');
		}
		$PHPCAS_CLIENT->setExtraCurlOption($key, $value);
		phpCAS :: traceEnd();
	}
```

```
<?php
/** auth_ldap_sync_users.php

 *  Modified for cas Module
 *
 *  This script is meant to be called from a cronjob to sync moodle with the LDAP
 *  backend in those setups where the LDAP backend acts as 'master'.
 *
* Recommended cron entry:
 *  # 5 minutes past 4am
 *  5 4
 *  * * /usr/bin/php -c /etc/php4/cli/php.ini
	/var/www/moodle/auth/ldap/auth_ldap_sync_users.php
 *
* Notes:
 *  - If you have a large number of users, you may want to raise the memory limits
 *  by passing -d momory_limit=256M
 *  - For debugging & better logging, you are encouraged to use in the command line:
 *  -d log_errors=1 -d error_reporting=E_ALL
	-d display_errors=0
	-d html_errors=0
 ** Performance notes:
 *  We have optimized it as best as we could for Postgres and mySQL, with 27K students
 *  we have seen this take 10 minutes.
 *
* /
```

```
if (isset($_SERVER['REMOTE_ADDR']))
        {
        error_log("should not be called from web server!");
        exit;
        }
        $nomoodlecookie = true;
        // cookie not needed
require_once(dirname(dirname(dirname(__FILE__))).'/config.php');
        // global moodle config file.
require_once($CFG->dirroot.'/course/lib.php');
                require_once($CFG->dirroot.'/lib/blocklib.php');
require_once($CFG->dirroot.'/mod/resource/lib.php');
                require_once($CFG->dirroot.'/mod/forum/lib.php');
                require_once($CFG->dirroot.'/lib/moodlelib.php');
        if (!is_enabled_auth('cas'))
        {
                echo "Plugin not enabled!";
   die;
        }
        $casauth = get_auth_plugin('cas');
        $casauth->sync_users(1000, true);
?>
```

```
/**
 * Email authentication plugin.
 */
class auth_plugin_email extends auth_plugin_base {

    /**
     * Constructor.
     */
    function auth_plugin_email() {
        $this->authtype = 'email';
        $this->config = get_config('auth/email');
    }

    /**
     * Returns true if the username and password work and false if they are
     * wrong or don't exist.
     *
     * @param string $username The username
     * @param string $password The password
     * @return bool Authentication success or failure.
     */
    function user_login ($username, $password) {
        global $CFG;
        if ($user = get_record('user', 'username', $username, 'mnethostid', $CFG-
>mnet_localhost_id)) {
            return validate_internal_user_password($user, $password);
        }
        return false;
    }
/**
     * Updates the user's password.
     *
     * called when the user password is updated.
     *
     * @param  object  $user        User table object  (with system magic quotes)
     * @param  string  $newpassword Plaintext password (with system magic quotes)
     * @return boolean result
     *
     */
    function user_update_password($user, $newpassword) {
        $user = get_complete_user_data('id', $user->id);
        return update_internal_user_password($user, $newpassword);
    }

    function can_signup() {
        return true;
    }

    /**
     * Sign up a new user ready for confirmation.
     * Password is passed in plaintext.
     *
     * @param object $user new user object (with system magic quotes)
     * @param boolean $notify print notice with link and terminate
     */
    function user_signup($user, $notify=true) {
        global $CFG;
        require_once($CFG->dirroot.'/user/profile/lib.php');

        $user->password = hash_internal_user_password($user->password);
```

```
        if (! ($user->id = insert_record('user', $user)) ) {
            print_error('auth_emailnoinsert','auth');
        }

        /// Save any custom profile field information
        profile_save_data($user);

        $user = get_record('user', 'id', $user->id);
        events_trigger('user_created', $user);

        if (! send_confirmation_email($user)) {
            print_error('auth_emailnoemail','auth');
        }

        if ($notify) {
            global $CFG;
            $emailconfirm = get_string('emailconfirm');
            $navlinks = array();
            $navlinks[] = array('name' => $emailconfirm, 'link' => null, 'type'
=> 'misc');
            $navigation = build_navigation($navlinks);
            print_header($emailconfirm, $emailconfirm, $navigation);
            notice(get_string('emailconfirmsent', '', $user->email), "$CFG-
>wwwroot/index.php");
        } else {
            return true;
        }
    }

    /**
     * Returns true if plugin allows confirming of new users.
     *
     * @return bool
     */
    function can_confirm() {
        return true;
    }

    /**
     * Confirm the new user as registered.
     *
     * @param string $username (with system magic quotes)
     * @param string $confirmsecret (with system magic quotes)
     */
    function user_confirm($username, $confirmsecret) {
        $user = get_complete_user_data('username', $username);
        if (!empty($user)) {
            if ($user->confirmed) {
                return AUTH_CONFIRM_ALREADY;

            } else if ($user->auth != 'email') {
                return AUTH_CONFIRM_ERROR;

            } else if ($user->secret == stripslashes($confirmsecret)) {   //
They have provided the secret key to get in
                if (!set_field("user", "confirmed", 1, "id", $user->id)) {
                    return AUTH_CONFIRM_FAIL;
                }
```

```
                    if (!set_field("user", "firstaccess", time(), "id", $user-
>id)) {
                        return AUTH_CONFIRM_FAIL;
                    }
                    return AUTH_CONFIRM_OK;
                }
            } else {
                return AUTH_CONFIRM_ERROR;
            }
        }

        function prevent_local_passwords() {
            return false;
        }

        /**
         * Returns true if this authentication plugin is 'internal'.
         *
         * @return bool
         */
        function is_internal() {
            return true;
        }

        /**
         * Returns true if this authentication plugin can change the user's
         * password.
         *
         * @return bool
         */
        function can_change_password() {
            return true;
        }

        /**
         * Returns the URL for changing the user's pw, or empty if the default
can
         * be used.
         *
         * @return mixed
         */
        function change_password_url() {
            return ''; // use dafult internal method
        }
```

```
        if ((!empty($data['username']) and !empty($data['email'])) or
(empty($data['username']) and empty($data['email']))) {
            $errors['username'] = get_string('usernameoremail');
            $errors['email']    = get_string('usernameoremail');

        } else if (!empty($data['email'])) {
            if (!validate_email($data['email'])) {
                $errors['email'] = get_string('invalidemail');

            } else if (count_records('user', 'email', $data['email']) > 1) {
                $errors['email'] = get_string('forgottenduplicate');

            } else {
                if ($user = get_complete_user_data('email', $data['email']))
{
                    if (empty($user->confirmed)) {
                        $errors['email'] = get_string('confirmednot');
                    }
                }
                if (!$user and empty($CFG->protectusernames)) {
                    $errors['email'] = get_string('emailnotfound');
                }
            }

        } else {
            if ($user = get_complete_user_data('username',
$data['username'])) {
                if (empty($user->confirmed)) {
                    $errors['email'] = get_string('confirmednot');
                }
            }
            if (!$user and empty($CFG->protectusernames)) {
                $errors['username'] = get_string('usernamenotfound');
            }
        }

        return $errors;
    }

}

?>
```

```
    if (!file_exists('./config.php')) {
        header('Location: install.php');
        die;
    }

    require_once('config.php');
    require_once($CFG->dirroot .'/course/lib.php');
    require_once($CFG->dirroot .'/lib/blocklib.php');

    if (empty($SITE)) {
        redirect($CFG->wwwroot .'/'. $CFG->admin .'/index.php');
    }

    // Bounds for block widths
    // more flexible for theme designers taken from theme config.php
    $lmin = (empty($THEME->block_l_min_width)) ? 100 : $THEME->block_l_min_width;
    $lmax = (empty($THEME->block_l_max_width)) ? 210 : $THEME->block_l_max_width;
    $rmin = (empty($THEME->block_r_min_width)) ? 100 : $THEME->block_r_min_width;
    $rmax = (empty($THEME->block_r_max_width)) ? 210 : $THEME->block_r_max_width;

    define('BLOCK_L_MIN_WIDTH', $lmin);
    define('BLOCK_L_MAX_WIDTH', $lmax);
    define('BLOCK_R_MIN_WIDTH', $rmin);
    define('BLOCK_R_MAX_WIDTH', $rmax);

    // check if major upgrade needed - also present in login/index.php
    if ((int)$CFG->version < 2006101100) { //1.7 or older
        @require_logout();
        redirect("$CFG->wwwroot/$CFG->admin/");
    }
    // Trigger 1.9 accesslib upgrade?
    if ((int)$CFG->version < 2007092000
        && isset($USER->id)
        && is_siteadmin($USER->id)) { // this test is expensive, but is only
triggered during the upgrade
        redirect("$CFG->wwwroot/$CFG->admin/");
    }

    if ($CFG->forcelogin) {
        require_login();
    } else {
        user_accesstime_log();
    }

    if ($CFG->rolesactive) { // if already using roles system
        if (has_capability('moodle/site:config',
get_context_instance(CONTEXT_SYSTEM))) {
            if (moodle_needs_upgrading()) {
                redirect($CFG->wwwroot .'/'. $CFG->admin .'/index.php');
            }
        } else if (!empty($CFG->mymoodleredirect)) {    // Redirect logged-in
users to My Moodle overview if required
            if (isloggedin() && $USER->username != 'guest') {
                redirect($CFG->wwwroot .'/my/index.php');
            }
        }
    } else { // if upgrading from 1.6 or below
        if (isadmin() && moodle_needs_upgrading()) {
            redirect($CFG->wwwroot .'/'. $CFG->admin .'/index.php');
        }
    }
```

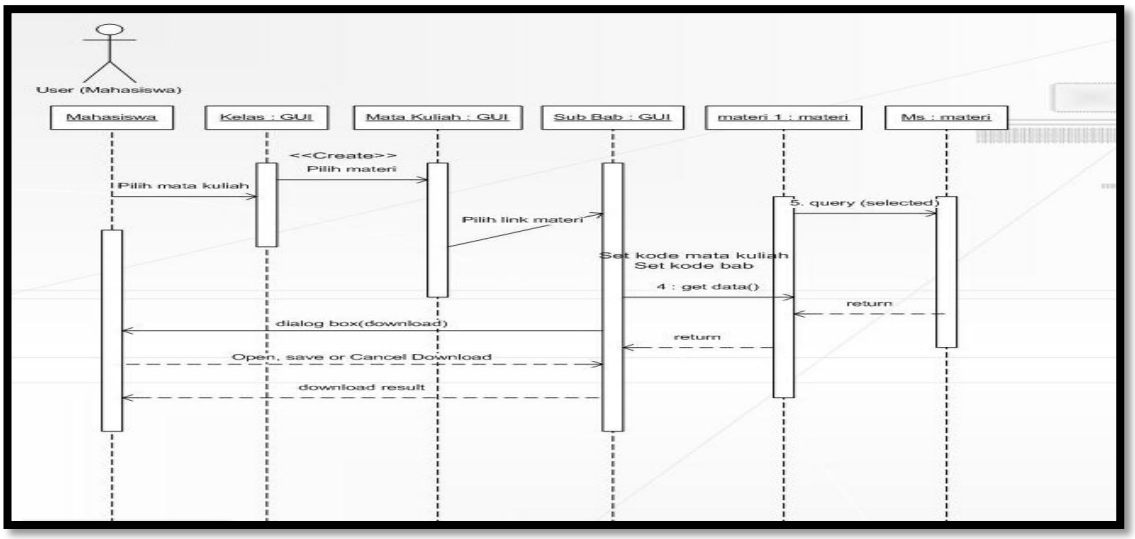
Perancangan sistem ini menggunakan UML (*Unified Modelling Languange*) sehingga standar untuk merancang model sebuah sistem dapat dibuat. Untuk itu terdapat beberapa tahapan dalam membuat sistem adalah sebagai berikut :

1. Sequence diagram

2.1 Sequence Diagram *User* Mendownload Materi Kuliah :

Sequence diagram dibawah ini menggambarkan skenario atau langkah-langkah yang dilakukan oleh *user* yaitu mahasiswa dalam men*download* materi kuliah.

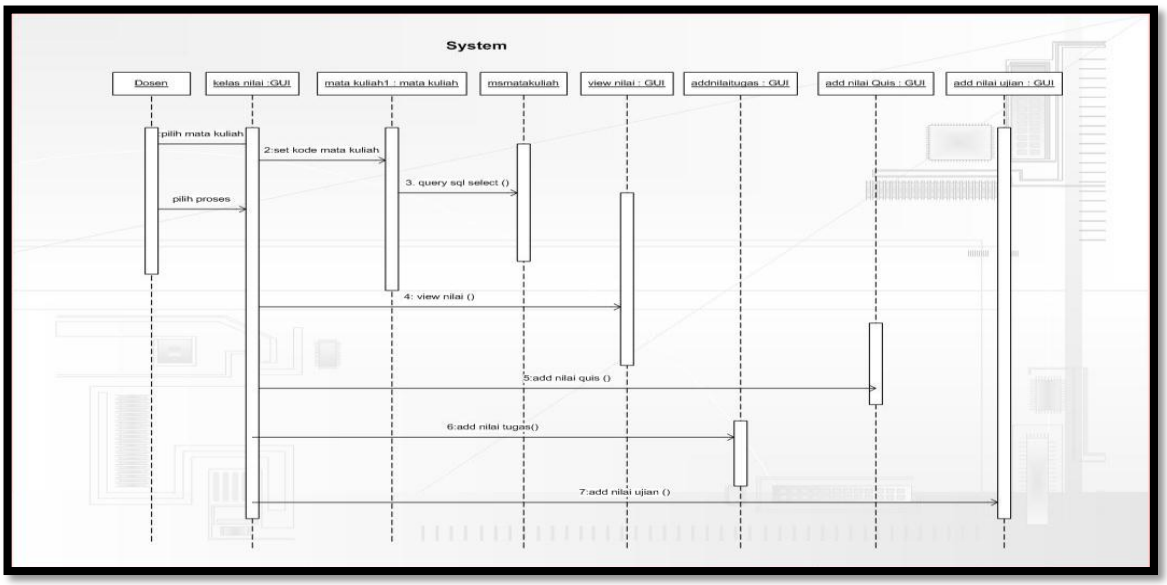
Rancangan *sequence* diagram untuk mahasiswa dalam men*download* materi kuliah dapat dilihat seperti pada gambar dibawah ini :



Gambar 1 *Sequence* mahasiswa *download* materi kuliah

2.2 Sequence diagram Dosen menginput nilai

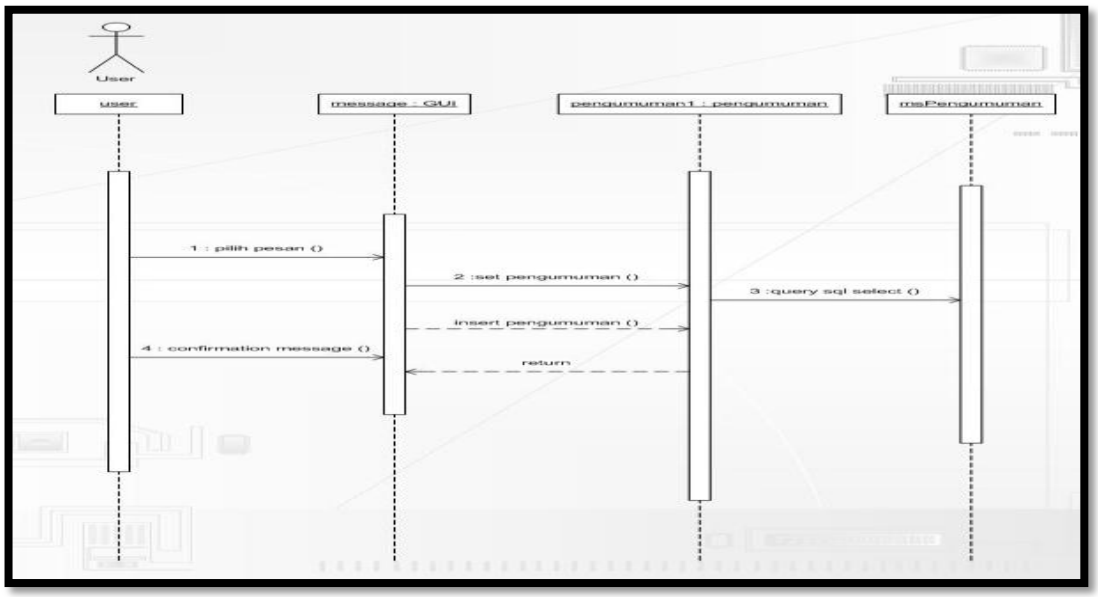
Sequence diagram dibawah ini menggambarkan skenario atau langkah-langkah yang dilakukan oleh *user* yaitu Dosen dalam menginput nilai. Rancangan *sequence* diagram untuk Dosen dalam menginput nilai dapat dilihat seperti pada gambar dibawah ini :



Gambar 2 *Sequence* Dosen input nilai

2.3 Sequence Diagram *User* Posting Pengumuman

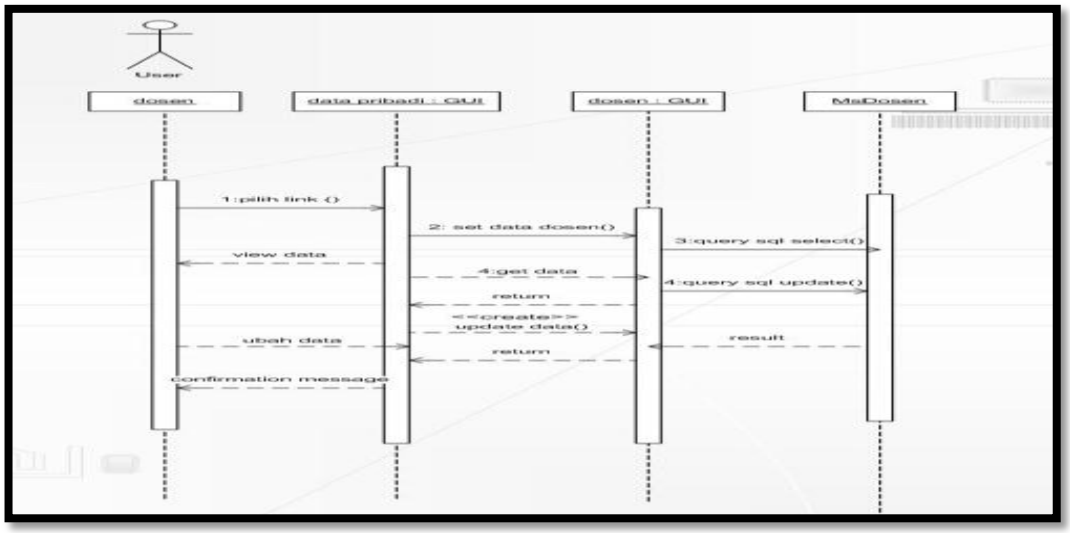
Sequence diagram dibawah ini menggambarkan skenario atau langkah-langkah yang dilakukan oleh *user* yaitu Dosen dan mahasiswa dalam posting pengumuman. Rancangan *sequence* diagram untuk *user* dalam posting pengumuman dapat dilihat seperti pada gambar dibawah ini :



Gambar 4.5 *Sequence user* posting pengumuman

1.4 Sequence Diagram Dosen *Update* Data Profil

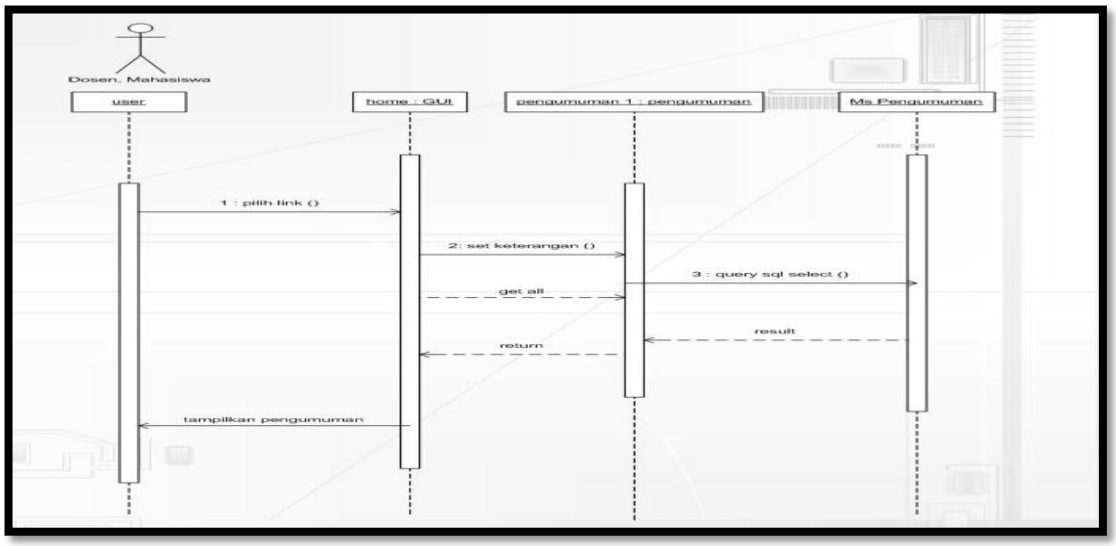
Sequence diagram dibawah ini menggambarkan skenario atau langkah-langkah yang dilakukan oleh *user* yaitu Dosen dalam meng*update* data profil. Rancangan *sequence* diagram untuk *user* dalam meng*update* profil dapat dilihat seperti pada gambar dibawah ini :



Gambar 3 *Sequence* Dosen *update* profil

2.5 Sequence Diagram *User* Melihat Pengumuman

Sequence diagram dibawah ini menggambarkan skenario atau langkah-langkah yang dilakukan oleh *user* yaitu Dosen dan mahasiswa dalam melihat pengumuman. Rancangan *sequence* diagram untuk *user* dalam melihat pengumuman dapat dilihat seperti pada gambar 2.5 dibawah ini :

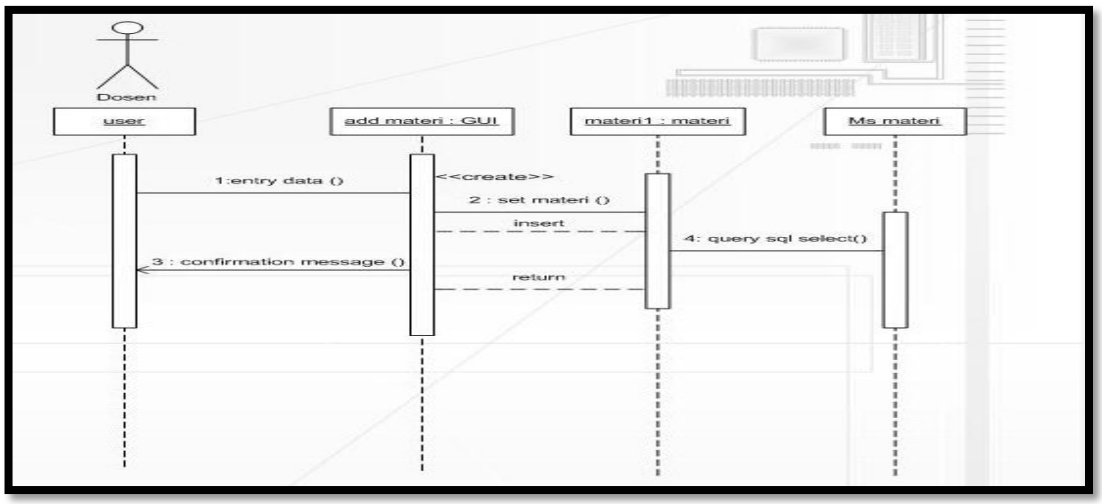


Gambar 4 *Sequence user* Mahasiswa lihat pengumuman

2.6 Sequence Diagram User Tambah Materi Kuliah

Sequence diagram dibawah ini menggambarkan skenario atau langkah-langkah yang dilakukan oleh user yaitu Dosen dalam menambah materi

kuliah. Rancangan *sequence* diagram untuk *user* Dosen dalam menambah materi kuliah dapat diihat seperti pada gambar dibawah ini :

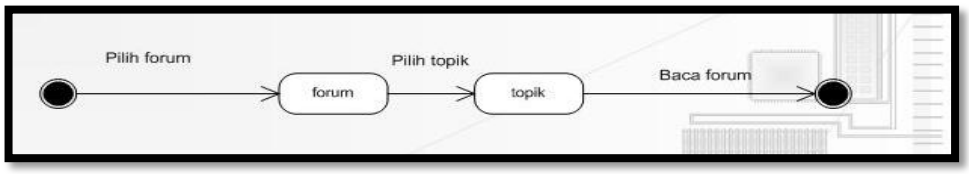


Gambar 5 *Sequence* Dosen tambah materi kuliah

2. Statechart Diagram

3.1 Statechart Baca Forum Diskusi

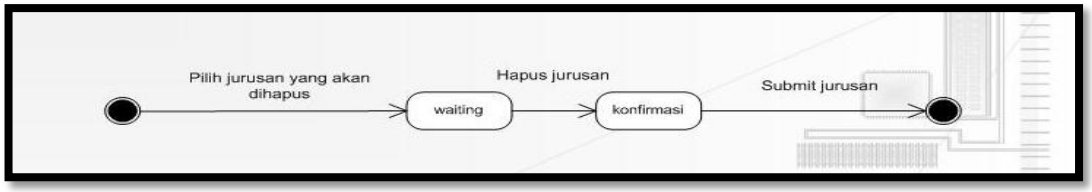
Statechart dibawah ini menggambarkan transisi dan perubahan keadaan baca forum diskusi. Rancangan statechart baca forum diskusi dapat dilihat seperti pada gambar dibawah ini :



Gambar 6 *Statechart* baca forum diskusi

3.2 Statechart Admin Hapus Kelas

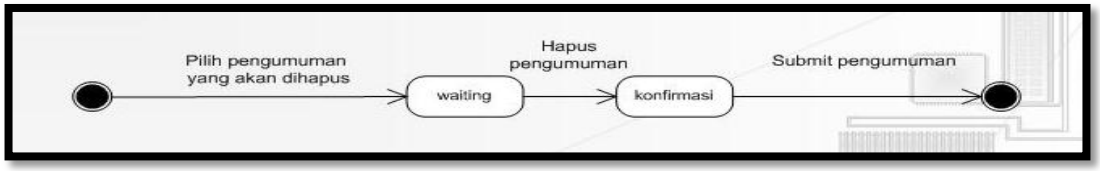
Statechart dibawah ini menggambarkan transisi dan perubahan keadaan admin hapus kelas. Rancangan *statechart* admin hapus kelas dapat dilihat seperti pada gambar dibawah ini :



Gambar 7 *Statechart* admin hapus kelas

3.3 Statechart Admin Hapus Pengumuman

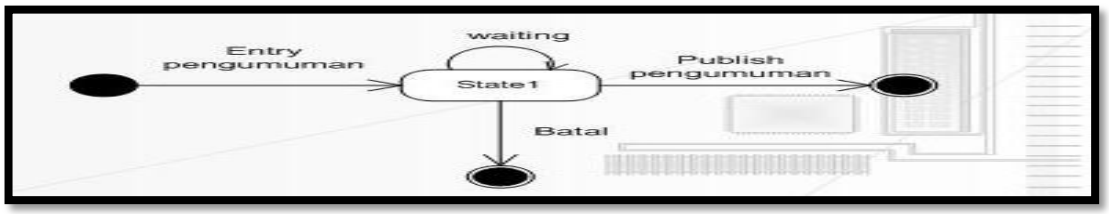
Statechart dibawah ini menggambarkan transisi dan perubahan keadaan admin hapus pengumuman. Rancangan *statechart* admin hapus pengumuman dapat dilihat seperti pada gambar dibawah ini :



Gambar 8 *Statechart* admin hapus pengumuman

3.4 Statechart Admin Posting Pengumuman

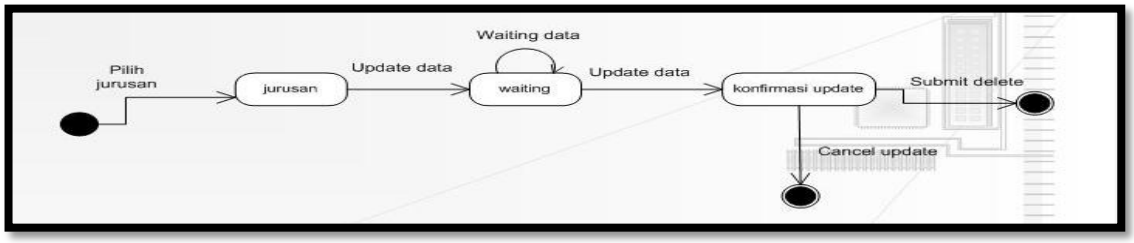
Statechart dibawah ini menggambarkan transisi dan perubahan keadaan admin posting pengumuman. Rancangan *statechart* admin posting pengumuman dapat dilihat seperti pada gambar dibawah ini :



Gambar 9 Admin posting pengumuman

3.5 Statechart Admin Update Kelas

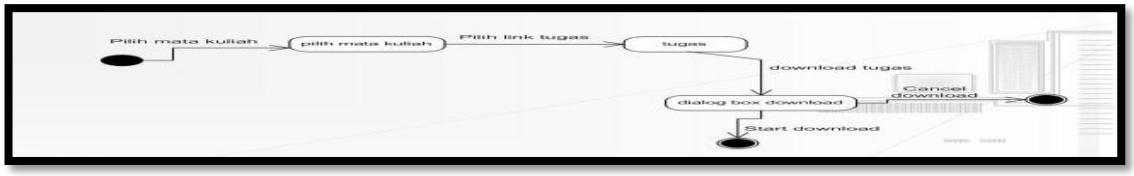
Statechart dibawah ini menggambarkan transisi dan perubahan keadaan admin update kelas. Rancangan *statechart* admin *update* kelas dapat dilihat seperti pada gambar dibawah ini :



Gambar 10 *Statechart* admin *update* kelas

3.6 Statechart *Download* Tugas Dan Materi Kuliah

Statechart dibawah ini menggambarkan transisi dan perubahan keadaan *user download* tugas dan materi kuliah. Rancangan *statechart user download* tugas dan materi kuliah dapat dilihat seperti pada gambar dibawah ini :

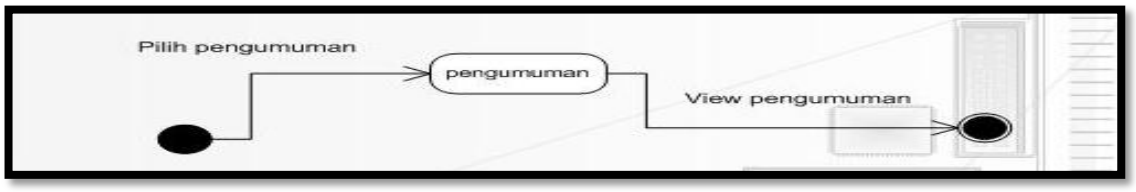


Gambar 11 Statechart download tugas dan materi kuliah

3.7 Statechart *View* Pengumuman

Statechart dibawah ini menggambarkan transisi dan perubahan keadaan user melihat pengumuman.

Rancangan *statechart user* melihat pengumuman dapat dilihat seperti pada gambar dibawah ini :

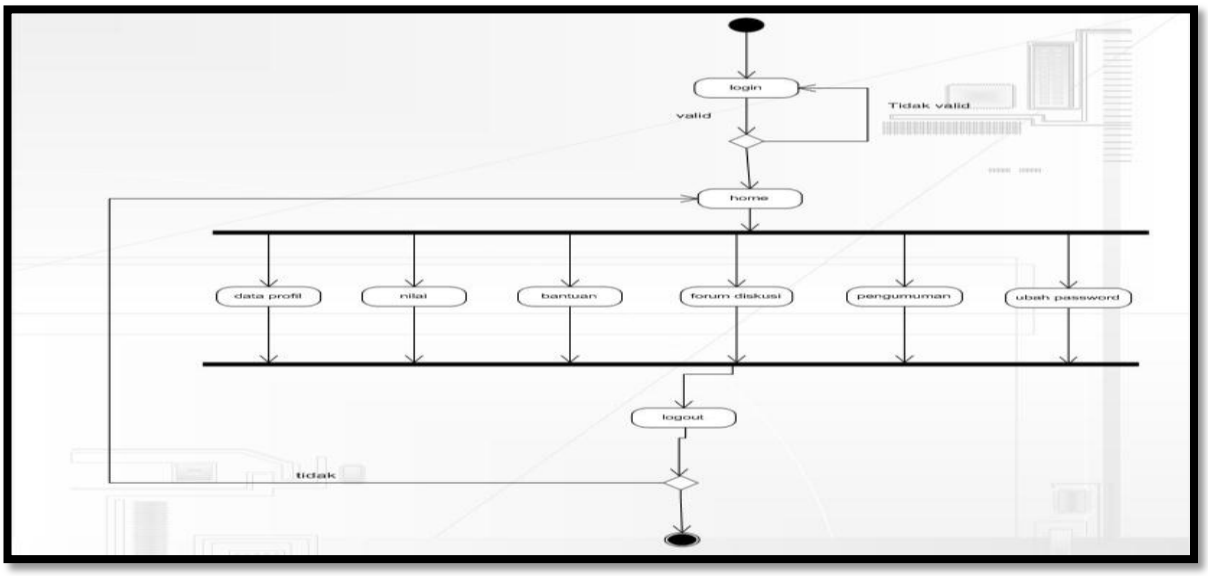


Gambar 12 *Statechart view* pengumuman

3. Activity Diagram

4.1 Activity Diagram Mahasiswa

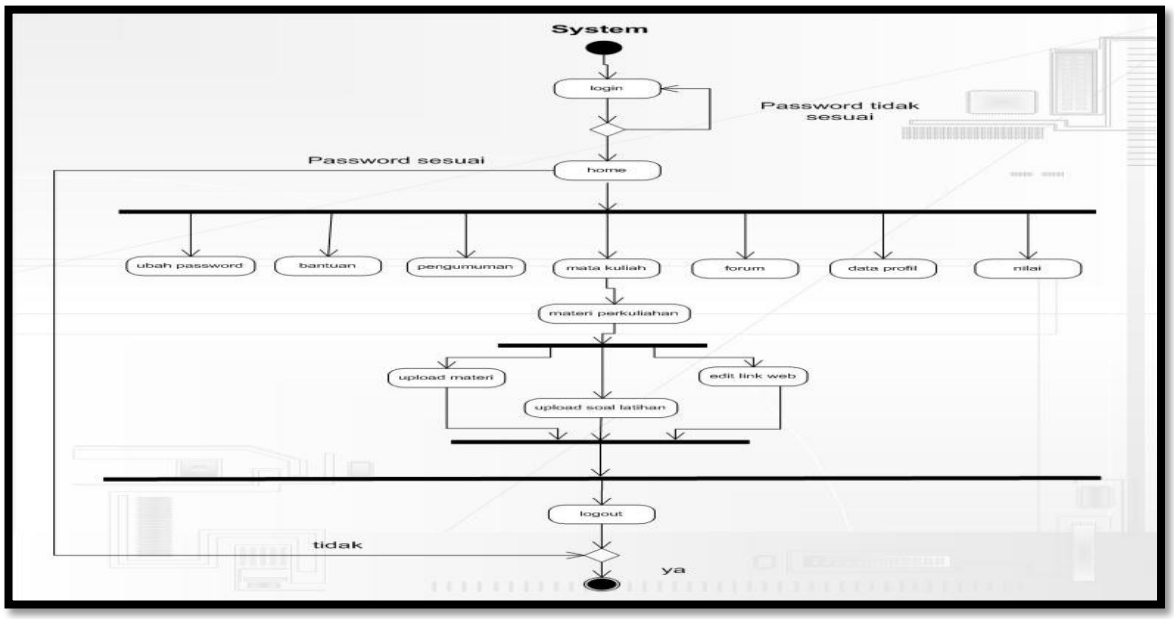
Activity diagram dibawah ini menggambarkan aliran aktivitas *user* yaitu Mahasiswa dalam menggunakan sistem untuk melakukan aktivitas. Rancangan *activity* diagram Mahasiswa dapat dilihat seperti pada gambar dibawah ini :



Gambar 13 *Activity* diagram Mahasiswa

4.2 Activity Diagram Dosen

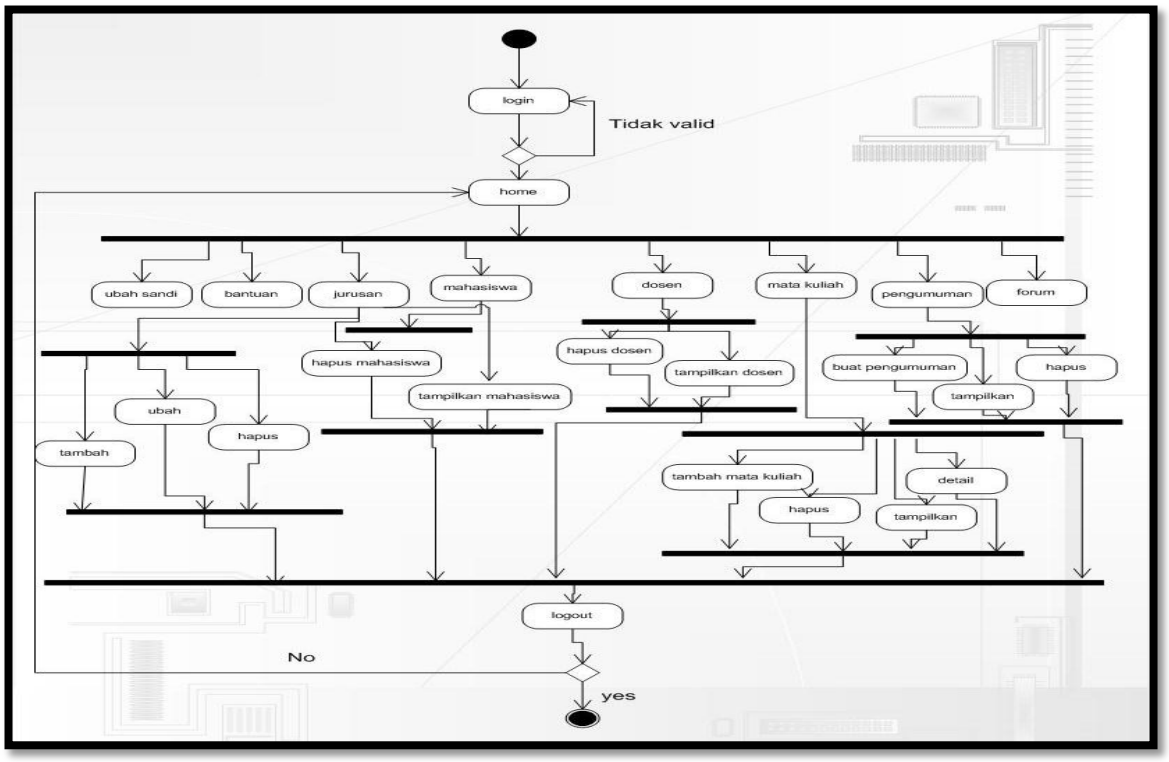
Activity diagram dibawah ini menggambarkan aktivitas user yaitu Dosen dalam menggunakan sistem untuk melakukan aktivitas. Rancangan *activity* diagram Dosen dapat dilihat seperti pada gambar dibawah ini :



Gambar 14 *Activity* diagram Dosen

4.3 Activity Diagram Admin

Activity diagram dibawah ini menggambarkan aktivitas *user* yaitu *admin* dalam menggunakan sistem untuk melakukan aktivitas. Rancangan *activity* diagram *admin* dapat dilihat seperti pada gambar dibawah ini :



Gambar 15 *Activity* diagram admin

4. Rancangan Antar Muka Sistem

Desain ini menggambarkan semua tampilan dari halaman-halaman situs *website pembelajaran non-line* yang akan dibuat. Pada halaman pertama kehalaman yang lain hanya dijalankan satu halaman yang ada pada index.php. akan tetapi halaman forum disamakan semua tampilannya kemudian tampilan halaman *admin* disamakan semua, agar pengguna dapat menggunakannya dengan mudah.

6.1 Rancangan Antar Muka Index

Halaman index menampilkan ucapan selamat datang untuk pengguna serta tampilan menu. Layar ini adalah layar awal yang pertama kali akan muncul ketika *website* ini diakses oleh *user*. Lambang dan nama perguruan tinggi diletakkan di bagian *banner*. Dibagian *banner* terdapat *link* untuk memasukkan *user name* serta *password* dan terdapat tombol untuk pengecekan *password* agar dapat masuk ke halaman berikutnya. Dibagian tengah terdapat link jurusan dan kelas

dan calenders sebagai info bagi pengguna. Dibagian *footer* terdapat hakcipta *E-learning* berbasis *web* perguruan tinggi. Rancangan halaman index dapat dilihat seperti pada gambar dibawah ini :

Gambar 16 Rancangan antar muka index

6.2 Rancangan Antar Muka *login*

Layar ini adalah layar *login*. Dibagian *banner* terdapat *link* untuk memasukkan *user name* serta *password* dan terdapat tombol untuk pengecekan *password* agar dapat masuk kehalaman berikutnya. Dibagian tengah terdapat *textbox* untuk memasukkan nama pengguna dan password. Rancangan halaman login dapat dilihat seperti pada gambar dibawah ini:

Gambar 17 Rancangan antar muka *login*

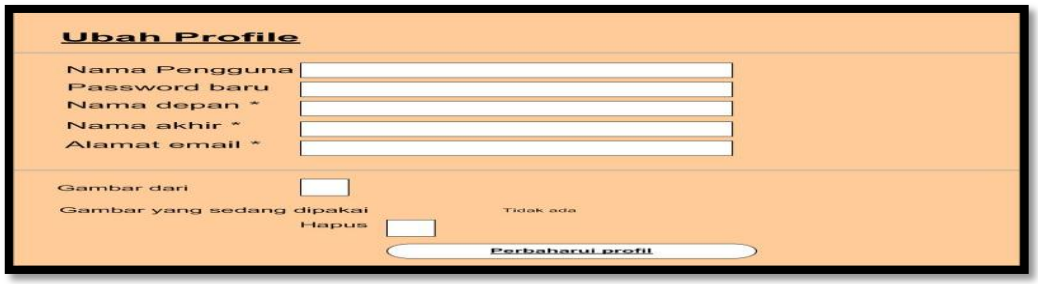
6.3 Rancangan Antar Muka *Admin*

Pada halaman ini user dapat memilih pilihan menu profil untuk melihat data profil. Rancangan halaman profil dapat dilihat seperti pada gambar dibawah ini:

Gambar 18 Rancangan antar muka *admin* profil

6.4 Rancangan Antar Muka Admin Ubah Profil

Pada halaman ini user dapat memilih pilihan menu ubah profil untuk mengubah data profil user. Rancangan halaman ubah profil dapat dilihat seperti pada gambar dibawah ini :

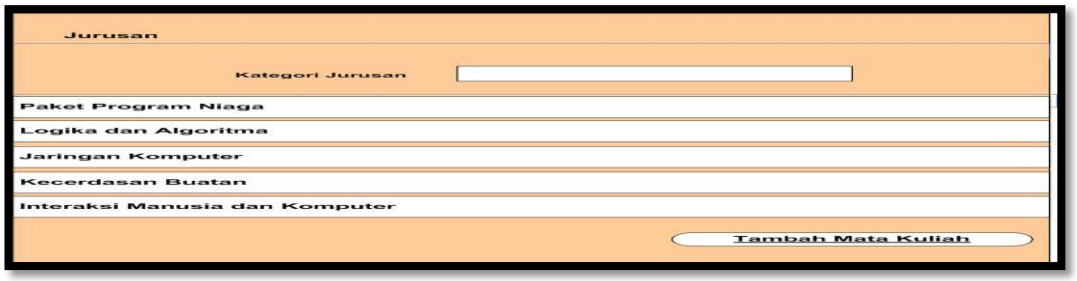


Gambar 19 Rancangan antar muka *admin* ubah profil

6.5 Rancangan Antar Muka Jurusan dan Kelas

Bila user memilih *link* jurusan dan kelas maka akan muncul halaman seperti pada gambar dibawah. Dalam halaman ini

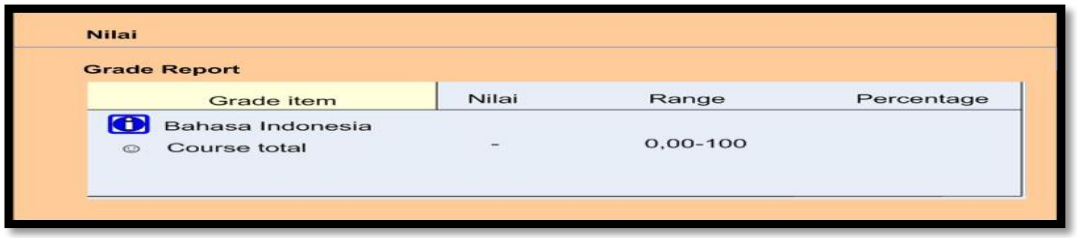
terdapat *link-link* yang merupakan kelas-kelas dan mata kuliah. *Link-link* tersebut dapat dipilih untuk mengakses berbagai kebutuhan dalam kegiatan belajar mengajar dikelas.



Gambar 20 Rancangan antar muka jurusan dan kelas

6.6 Rancangan Antar Muka Nilai

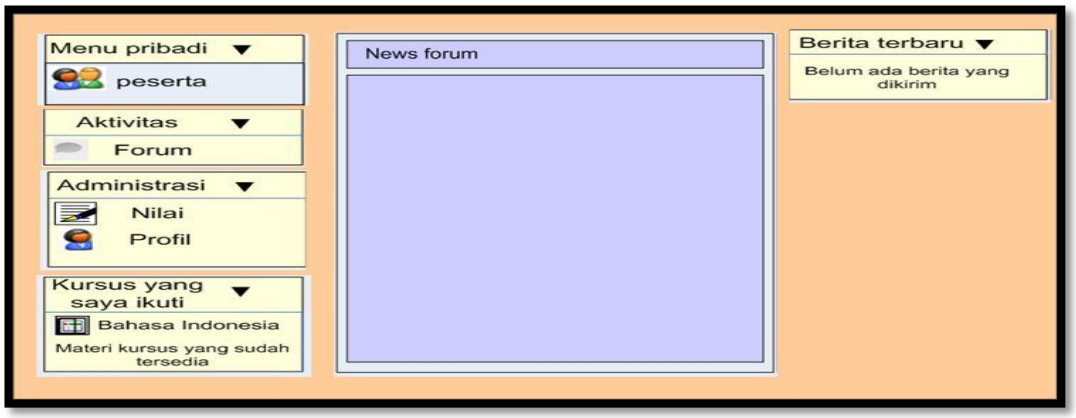
Bila *user* memilih *link* nilai, maka akan muncul halaman seperti pada gambar Dalam hal ini akan menampilkan kelas dimana Dosen akan mengajar. Ditampilkan nama-nama kelas beserta nama mata kuliahnya.



Gambar 21 Rancangan antar muka nilai

6.7 Rancangan Antar Muka Mata Kuliah

Bila user ingin melihat mata kuliah maka akan ditampilkan mata kuliah seperti pada gambar dibawah ini:



Gambar 22 Rancangan antar muka mata kuliah

6.8 Rancangan Antar Muka Forum

Bila *user* Dosen dan Mahasiswa ingin berinteraksi melalui Forum diskusi maka akan ditampilkan topik apa yang akan dibahas, maka akan ditampilkan dibawah ini :

Gambar 23 Antar muka forum

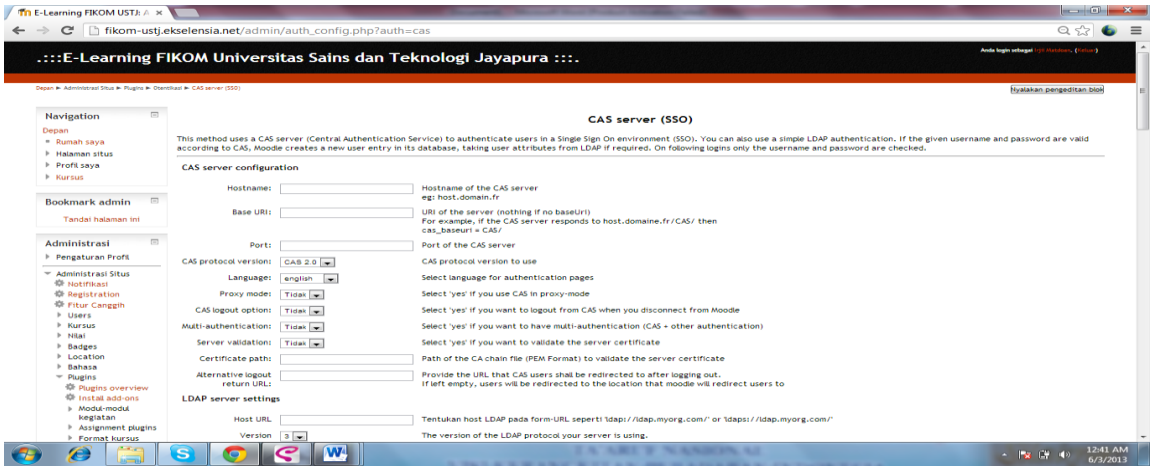
B. Implementasi Sistem Otentikasi

Seperti pada penjelasan di awal perancangan maka pada tahapan implementasi sistem otentikasi terdapat 2 cara untuk melakukan otentikasi adalah sebagai berikut :

1. Implementasi Sistem Otentikasi

   Pada pengaturan implementasi pada sistem ini dengan langkah-langkah adalah sebagai berikut :

   Memilih *Users* – *Authentication* – CAS *server*, maka tampilan antar muka seperti pada gambar dibawah ini :

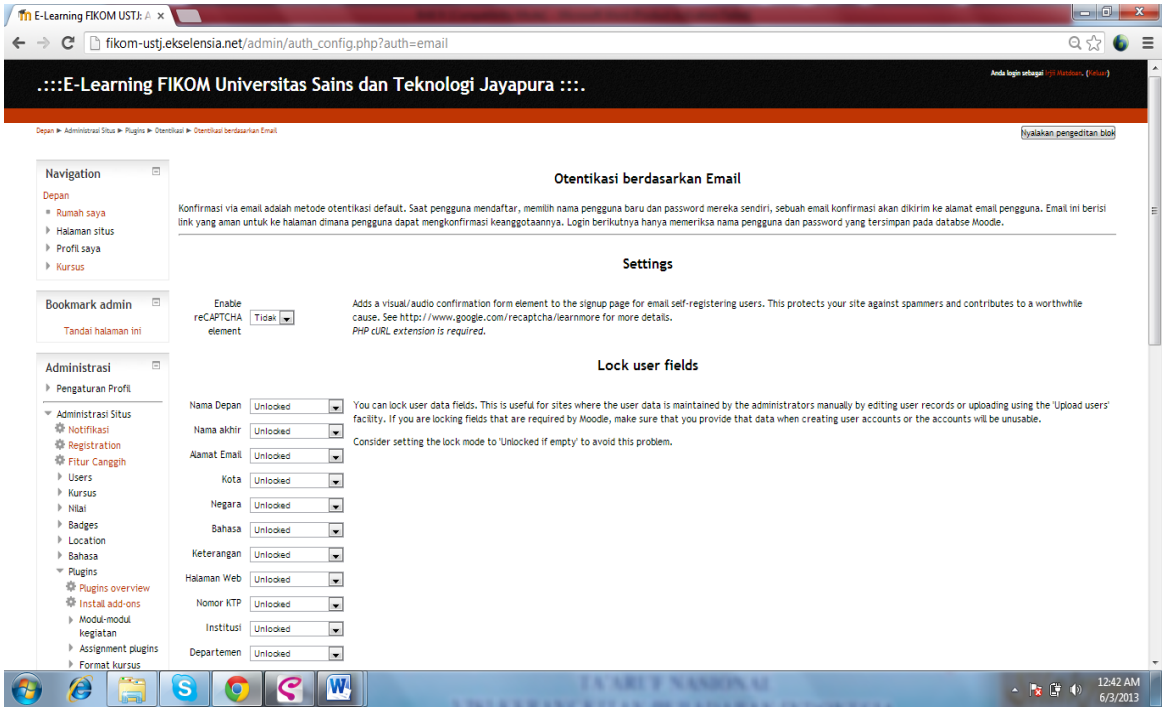


Gambar 24 Tampilan implementasi antar muka konfigurasi CAS server dan LDAP server

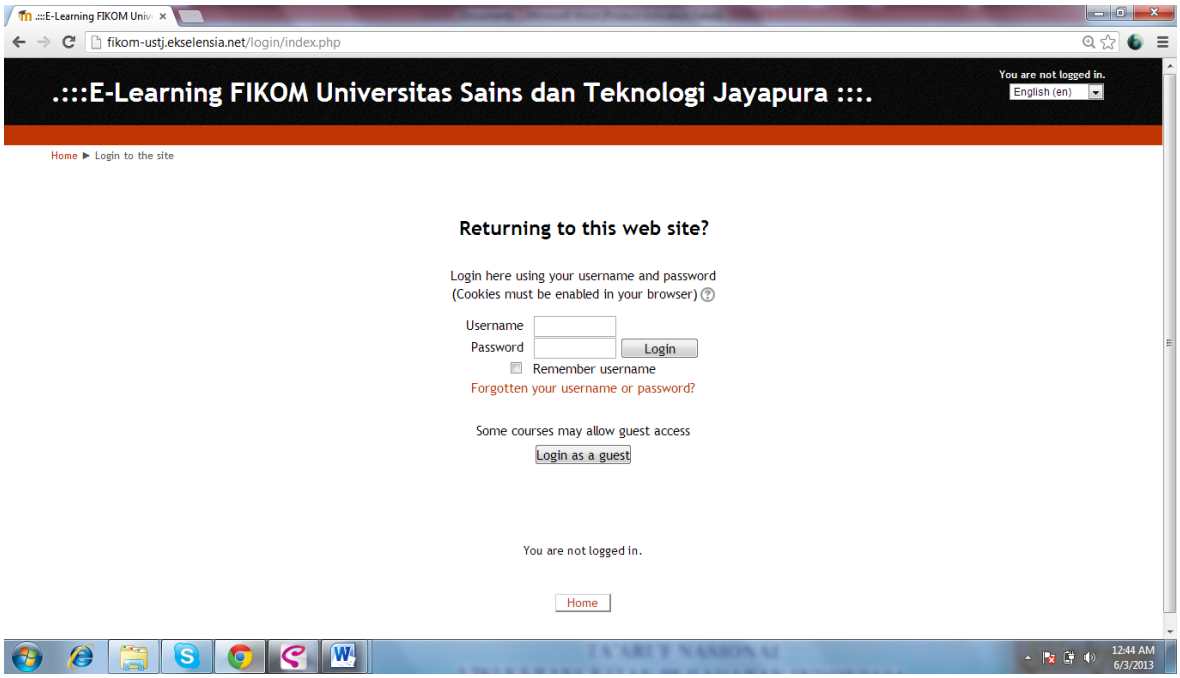
2. Implementasi Otentikasi Berdasarkan *email*

   Pada tahapan ini pengaturan otentikasi berdasarkan *email* dilakukan dengan cara adalah sebagai berikut:

   Memilih *Users* – *Authentication* – Otentikasi Berdasarkan *email*, maka tampilan antar muka seperti pada gambar dibawah ini :



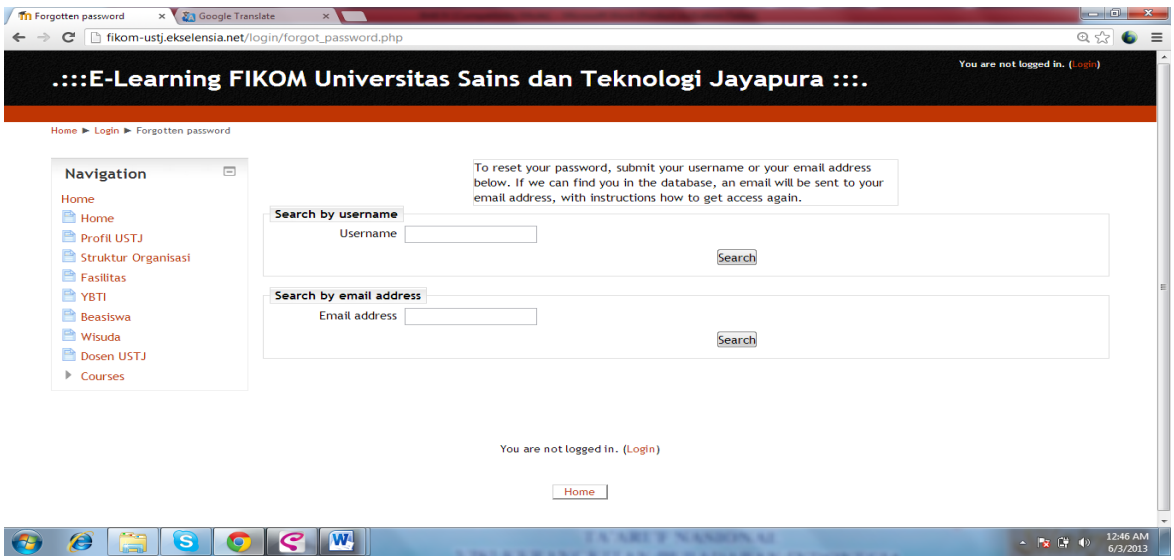
Gambar 25 Tampilan implementasi antar muka otentikasi berdasarkan *email*



Gambar 26 Hasil Implementasi Login CAS server

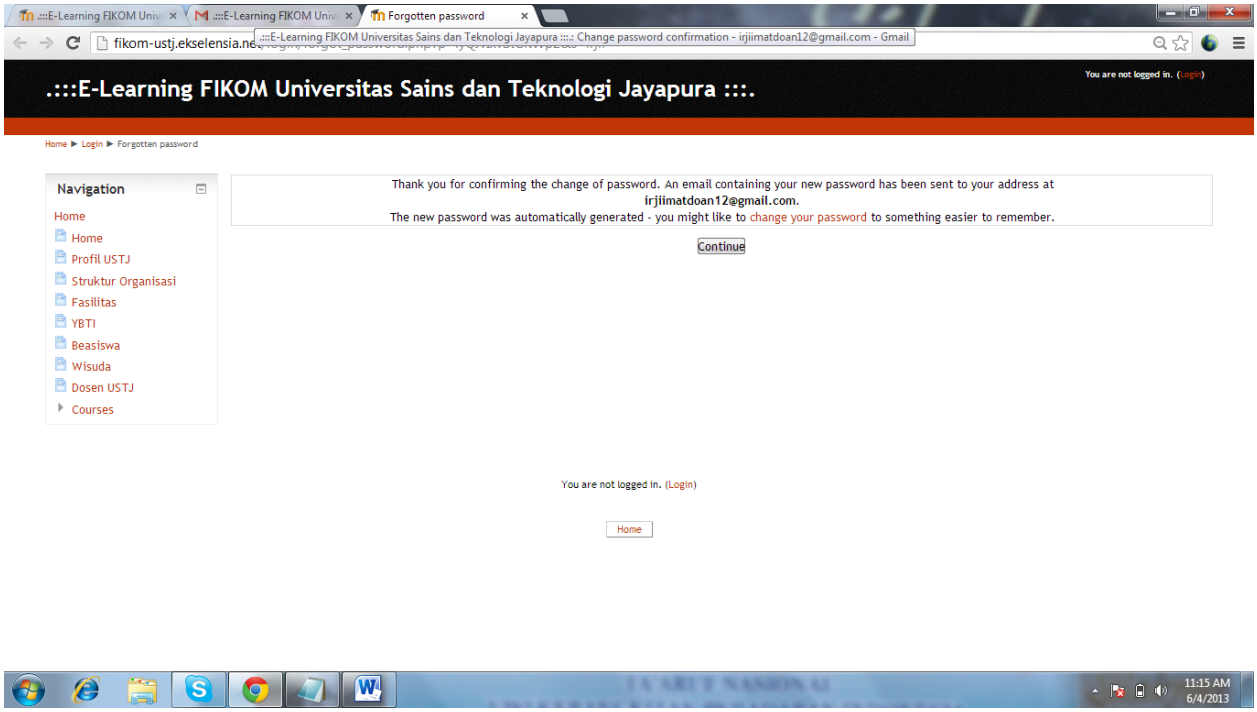
```
        if ((!empty($data['username']) and !empty($data['email'])) or
(empty($data['username']) and empty($data['email']))) {
            $errors['username'] = get_string('usernameoremail');
            $errors['email']    = get_string('usernameoremail');

        } else if (!empty($data['email'])) {
            if (!validate_email($data['email'])) {
                $errors['email'] = get_string('invalidemail');

            } else if (count_records('user', 'email', $data['email']) > 1) {
                $errors['email'] = get_string('forgottenduplicate');

            } else {
                if ($user = get_complete_user_data('email', $data['email']))
{
                    if (empty($user->confirmed)) {
                        $errors['email'] = get_string('confirmednot');
                    }
                }
                if (!$user and empty($CFG->protectusernames)) {
                    $errors['email'] = get_string('emailnotfound');
                }
            }

        } else {
            if ($user = get_complete_user_data('username',
$data['username'])) {
                if (empty($user->confirmed)) {
                    $errors['email'] = get_string('confirmednot');
                }
            }
            if (!$user and empty($CFG->protectusernames)) {
                $errors['username'] = get_string('usernamenotfound');
            }
        }

        return $errors;
    }

}

?>
```

Bila *username* dan *password* yang dimasukkan salah maka pengguna tidak bisa masuk ke halaman menu utama, dan pengguna pertama harus ditemukan dalam database sistem. Masukkan nama pengguna atau alamat *email* pengguna yang terdaftar dalam kotak yang sesuai. Tidak perlu untuk memasukkan keduanya.



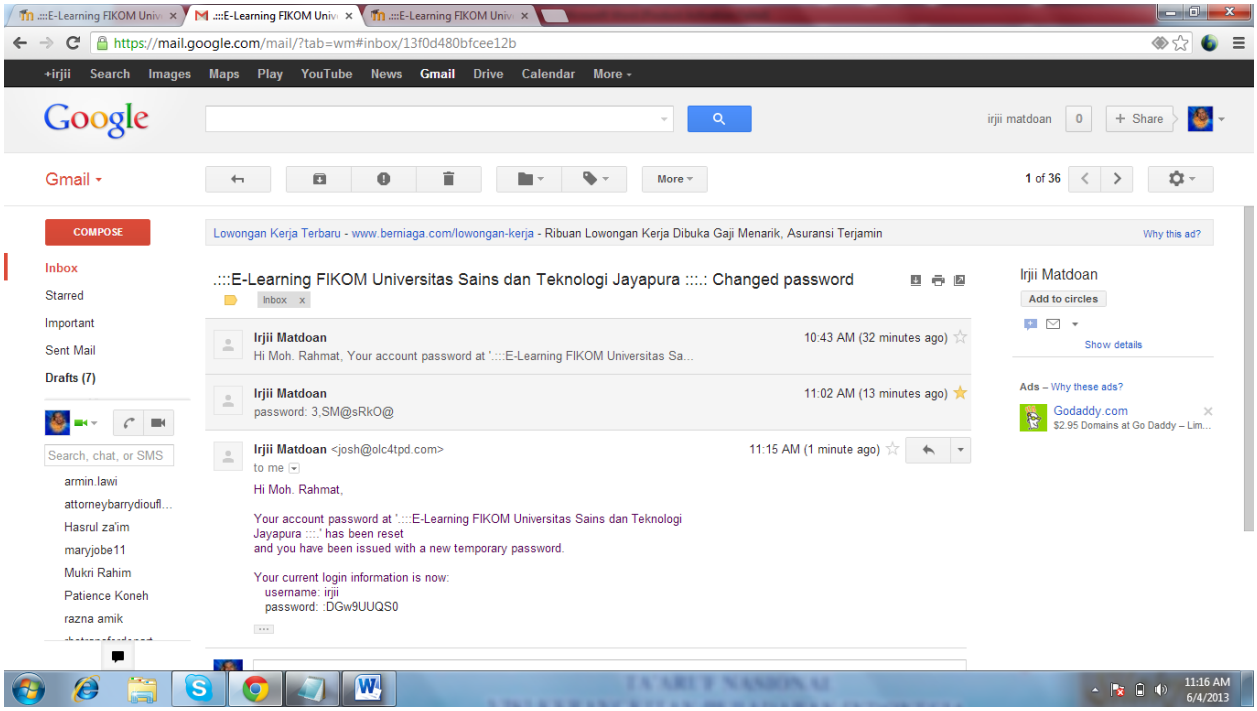
Gambar 27 Hasil Tampilan Konfirmasi bila *username* dan *password* yang terlupa

```
            print_header(get_string("confirmed"), get_string("confirmed"),
array(), "");
            print_box_start('generalbox centerpara boxwidthnormal
boxaligncenter');
            echo "<h3>".get_string("thanks").", ". fullname($USER) .
"</h3>\n";
            echo "<p>".get_string("confirmed")."</p>\n";
            print_single_button("$CFG->wwwroot/course/", null,
get_string('courses'));
            print_box_end();
            print_footer();
            exit;
        } else {
            error("Invalid confirmation data");
        }
    } else {
        print_error("errorwhenconfirming");
    }

    redirect("$CFG->wwwroot/");
```



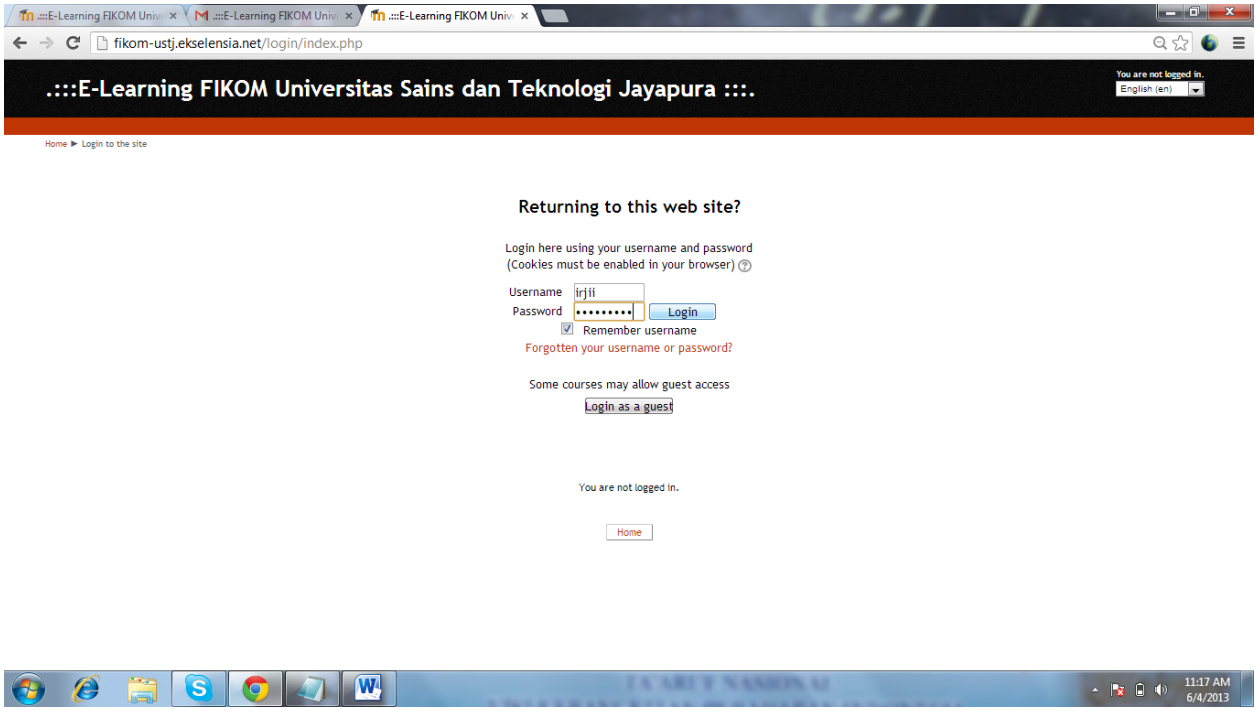
Gambar 28 Tampilan alamat *email* dari pengguna, untuk dikirimkan perubahan *password*

Selanjutnya jika kita akan melihat *password* yang telah dirubah, akan muncul pada alamat *email* pengguna



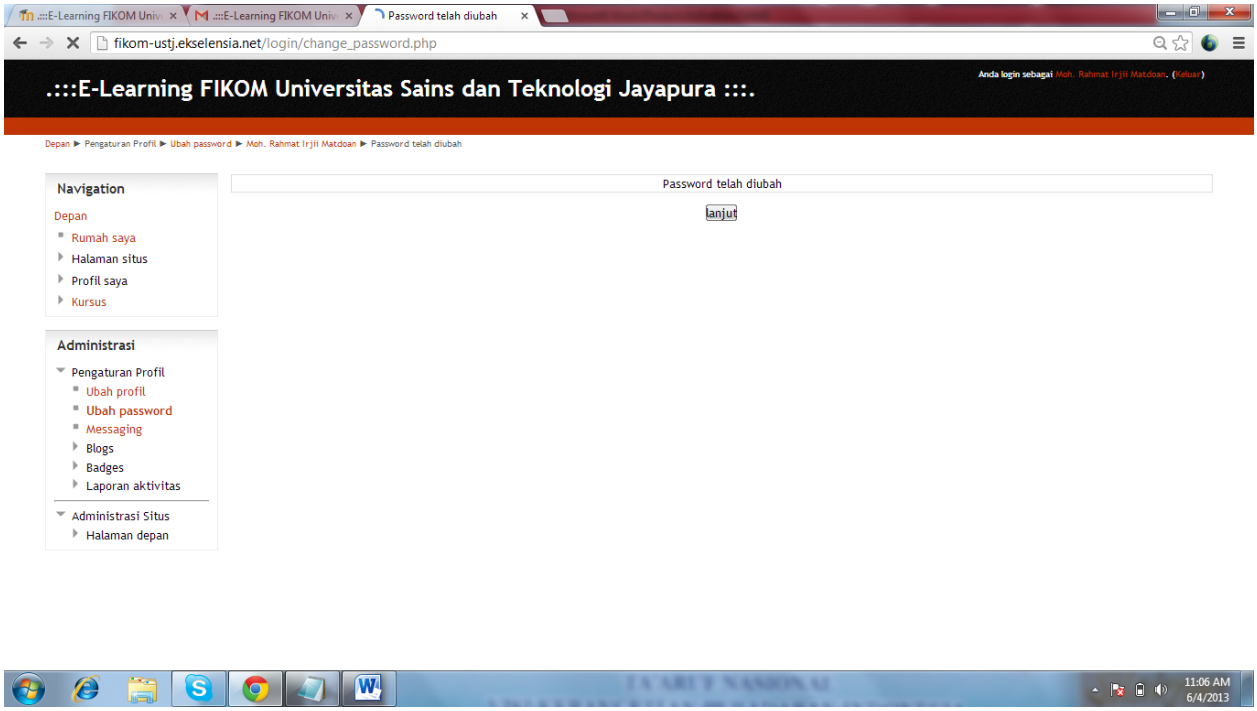
Gambar 29 Tampilan inbox email pengguna untuk perubahan *password*

Password yang telah terkirim melalui *email*, kemudian kita masukkan ke menu *login*



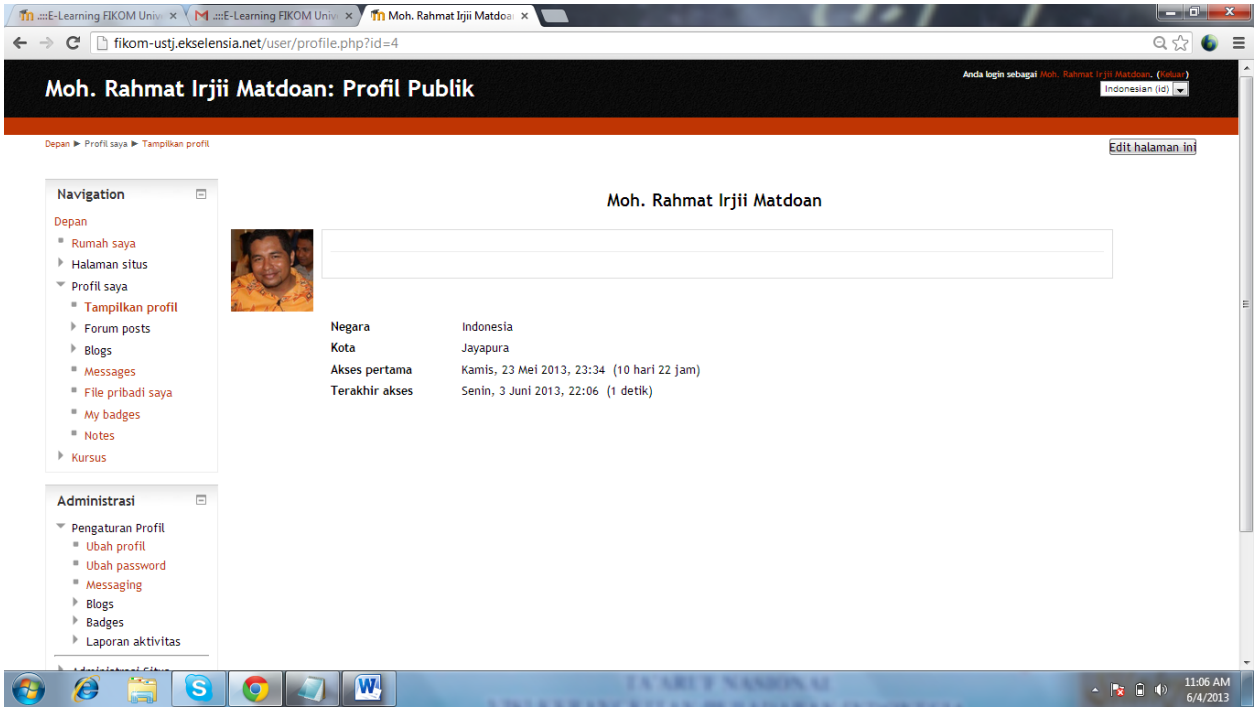
Gambar 30 Tampilan halaman login Otentikasi

Setelah kita masukkan *password*, maka pemberitahuan *password* telah diubah dan secara otomatis pengguna telah masuk pada Halaman *e-learning*

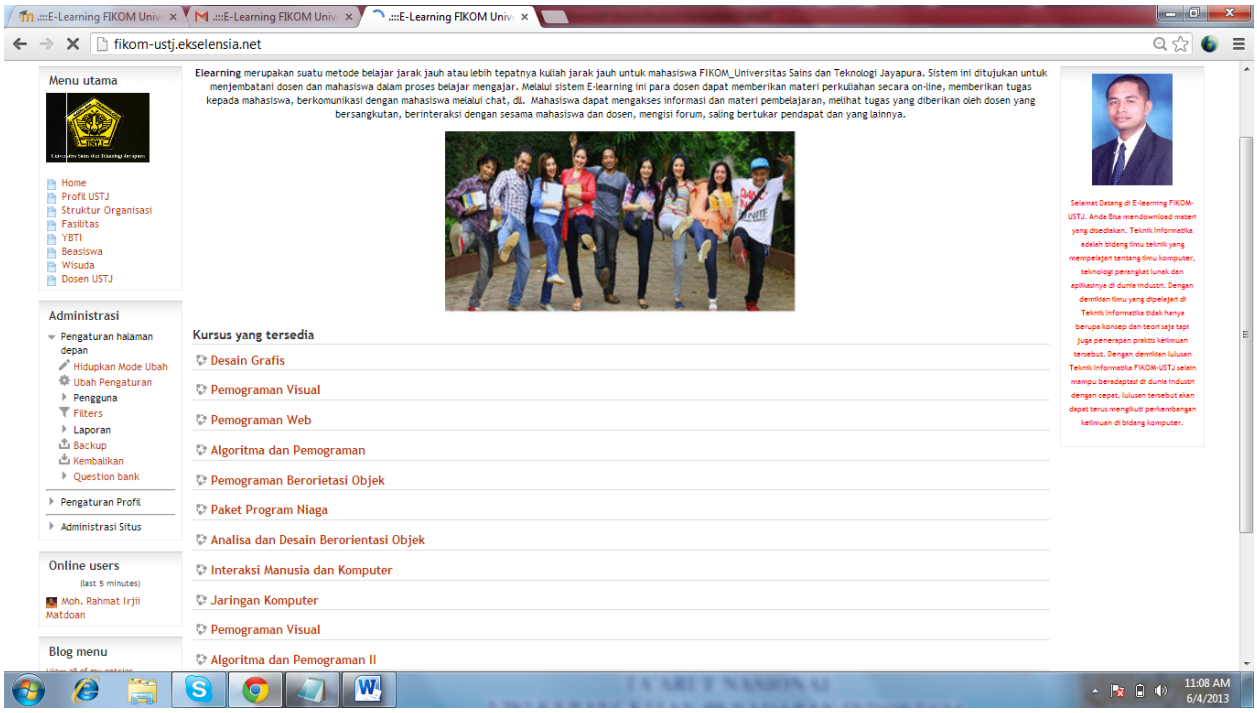


Gambar 31 Tampilan *password* yang telah diubah

Inilah hasil tampilannya setelah melalui proses otentikasi berdasarkan *email*.

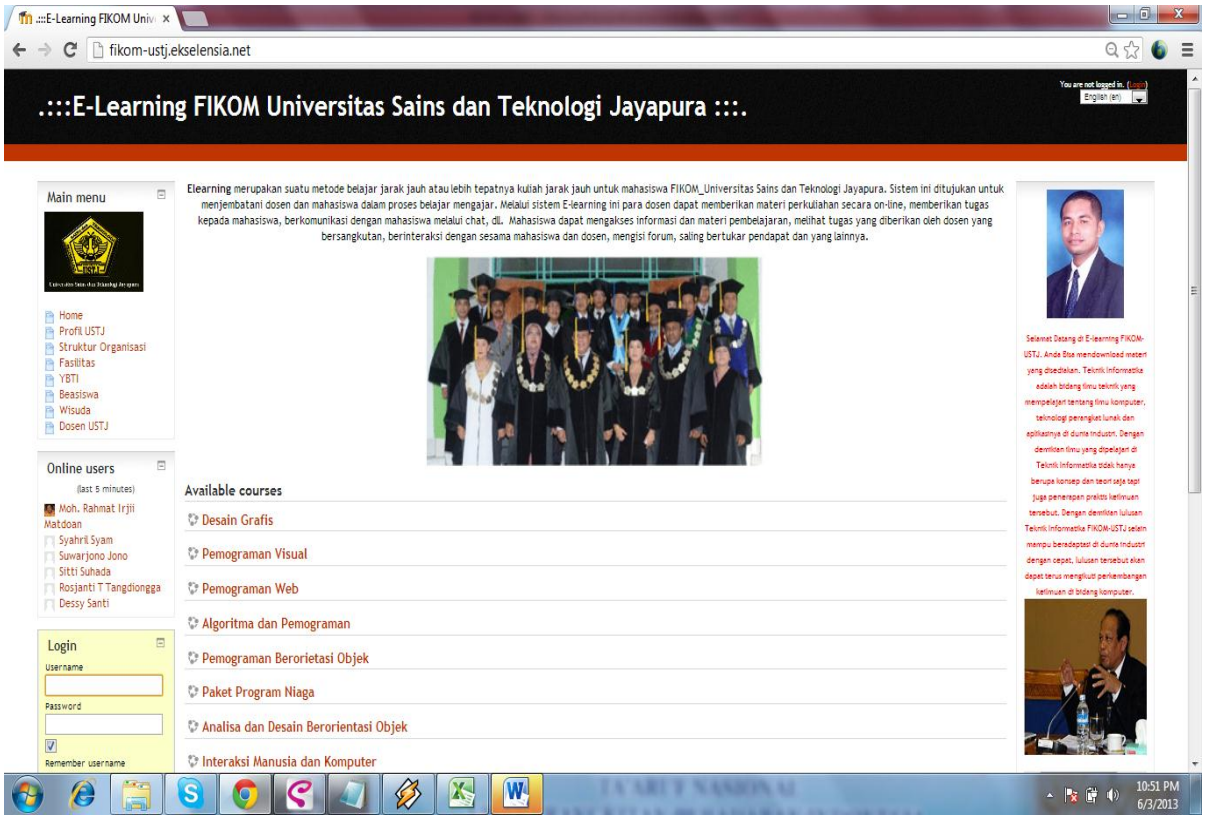


Gambar 32 Tampilan profil dari pengguna



Gambar 33 Tampilan halaman utama *e-learning*

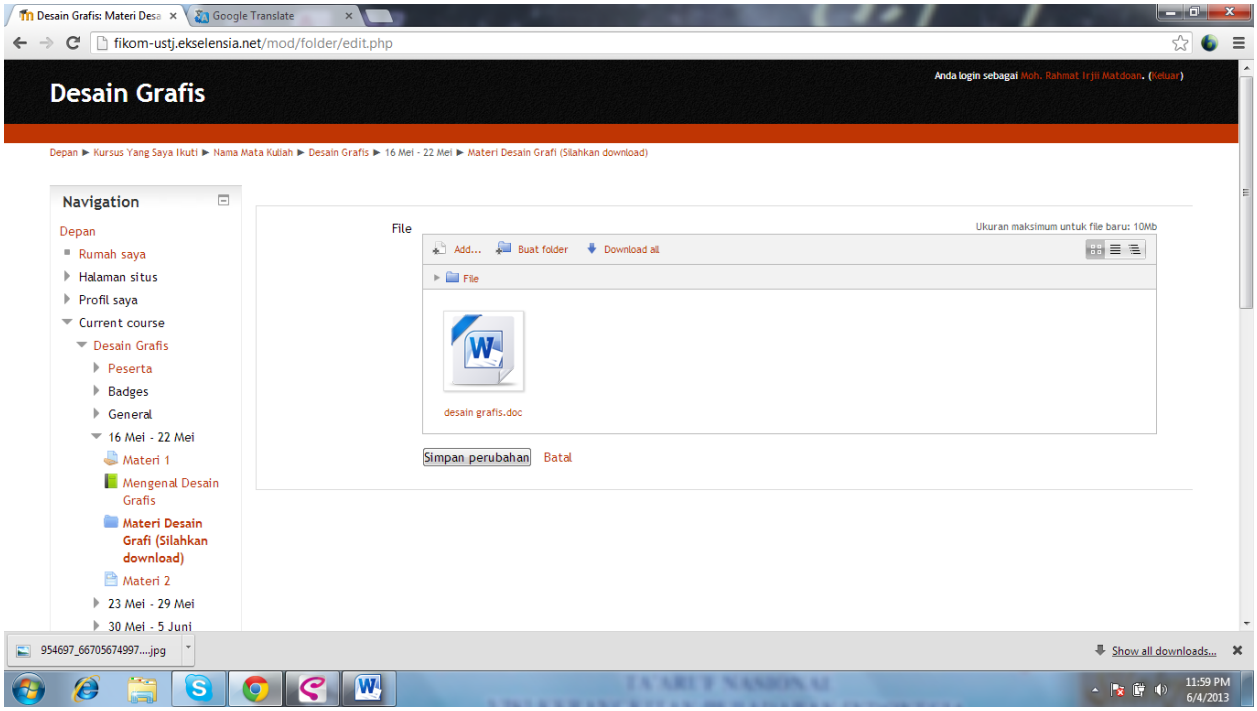
Halaman tampilan ketika beberapa *user* sedang aktif dalam menggunakan *e-learning* terlihat jelas pada menu *online users* seperti pada tampilan dibawah ini:



Gambar 34 Hasil tampilan ketika *user* sedang aktif menggunakan *e-learning*

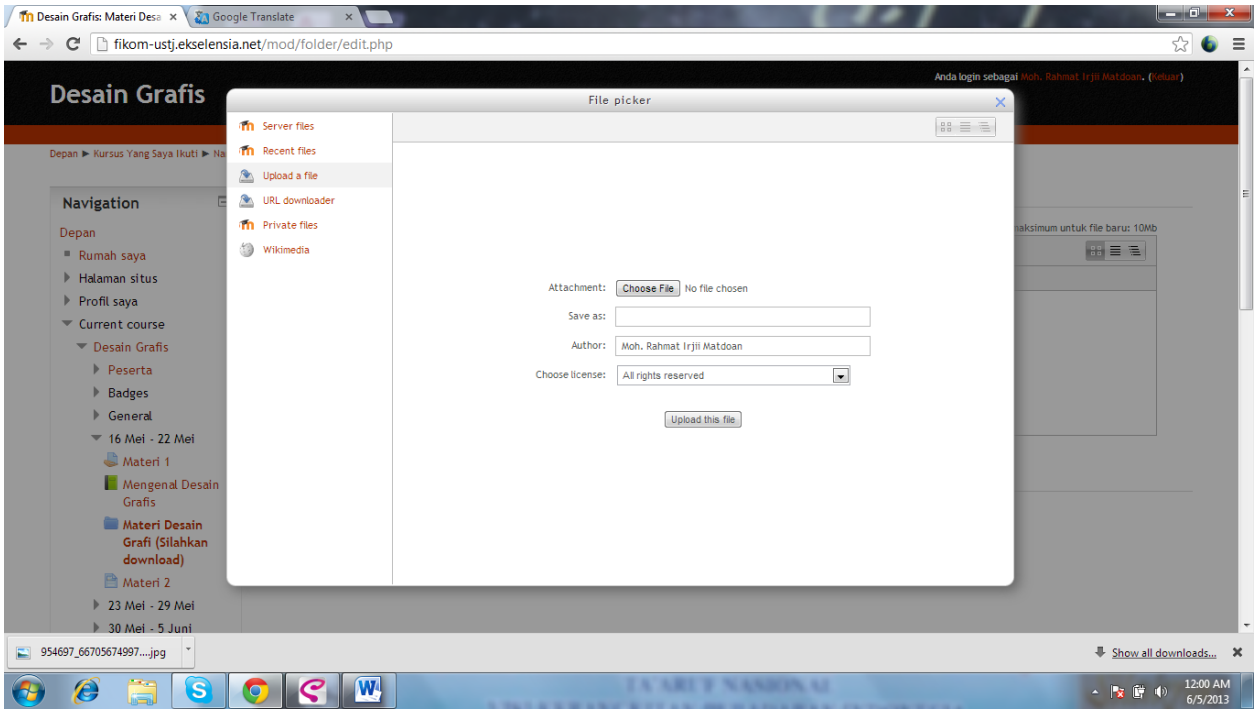
Langkah-langkah mengupload materi kuliah :

Pada materi kuliah Desain Grafis Pilih Add untuk mengupload materi



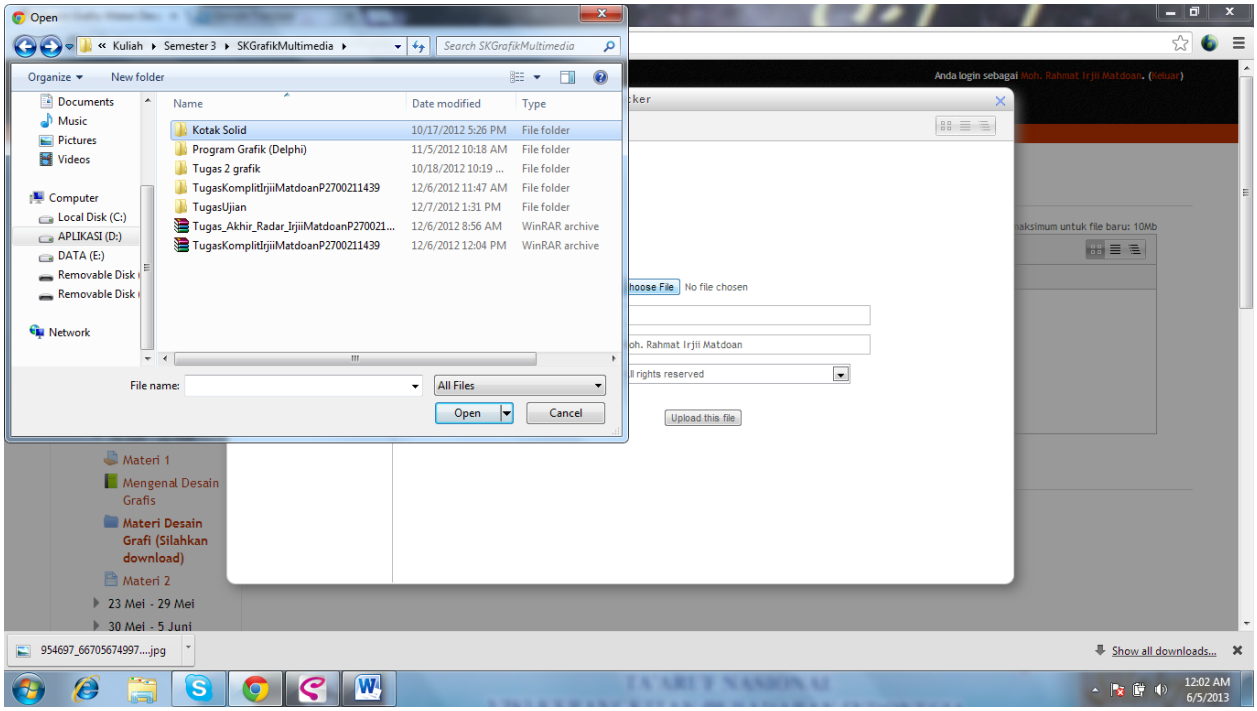
Gambar 35 Tampilan file yang akan diupload

Pada menu tampilan File picker, kemudian pilihlah upload a file



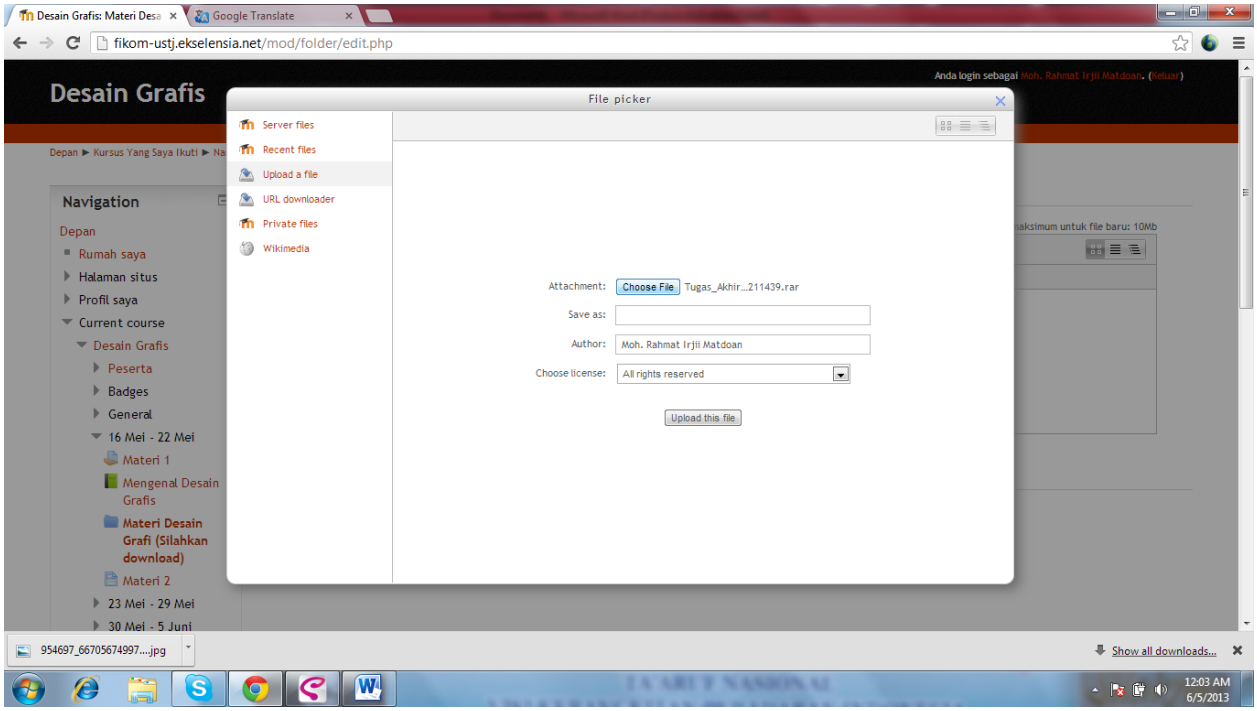
Gambar 36 Tampilan menu file picker

Setelah memilih upload a file, kemudian muncul kotak dialog file yang akan dipilih



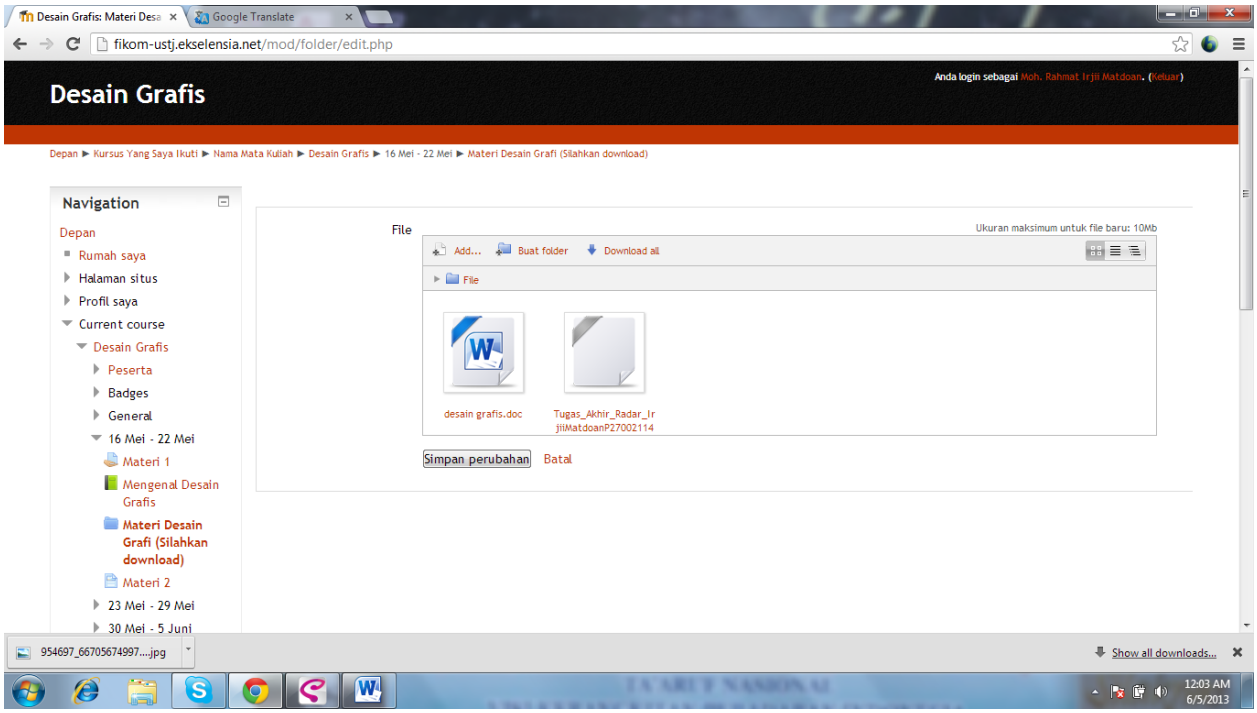
Gambar 37 Tampilan pilihan file yang akan diupload

Setelah memilih file yang akan diupload kemudian pada menu file picker telah ada file Tugas_akhir yang telah diupload, setelah itu tekan tombol upload file



Gambar 38 Tampilan file yang akan diupload

Pada materi desain grafis telah terlihat file yang sudah terupload kedalam sistem



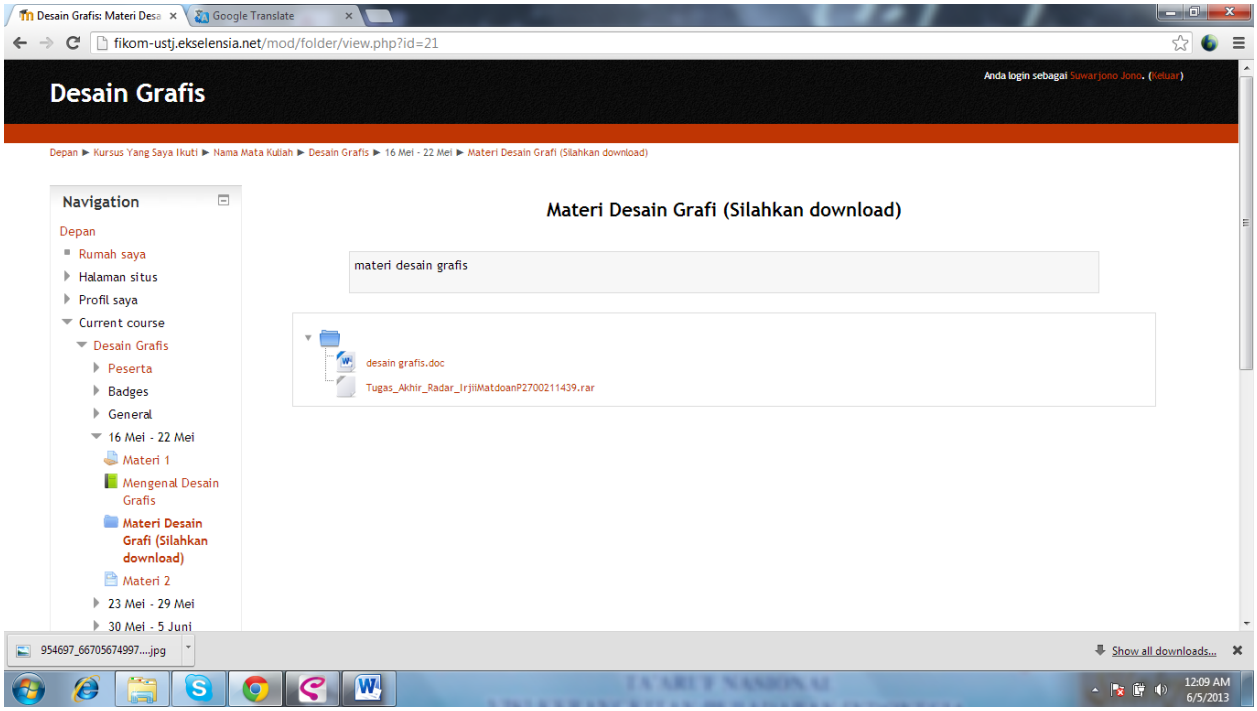
Gambar 39 Tampilan file telah diupload

Langkah-langkah mendownload materi:

Pilih materi Desain Grafis

Gambar 40 Tampilan pengguna yang akan mengakses *e-learning*

Pada materi desain grafis pengguna dapat mendownload materi tersebut, dengan memilih pada gambar dibawah ini:



Gambar 41 Tampilan materi yang akan didownload

Responden :
Hari/Tanggal :

Berilah penilaian untuk setiap pertanyaan dengan cara memberi tanda √ pada kolom Pendapat yang ada

No	Pertanyaan	Pendapat	
1	Apakah menurut anda sistem otentikasi *e-learning* melalui *Central Authentication Service (CAS)* berbasis *open source* berjalan sesuai dengan aplikasinya?	Sesuai Cukup Sesuai Kurang Sesuai Tidak Sesuai	[ ] [ ] [ ] [ ]
2	Bagaimana menurut anda apakah proses otentikasi *e-learning* sudah sesuai?	Sesuai Cukup Sesuai Kurang Sesuai Tidak Sesuai	[ ] [ ] [ ] [ ]
3	Bagaimana menurut anda apakah *interface* yang dibangun mudah untuk digunakan?	Mudah Cukup Kurang Susah	[ ] [ ] [ ] [ ]
4	Apakah anda setuju untuk dapat menggunakan *e-learning* sebagai media pembelajaran di fakultas FIKOM-USTJ secara *online*?	Sangat Setuju Setuju Kurang Setuju Tidak Setuju	[ ] [ ] [ ] [ ]
5	Apakah menurut anda fasilitas yang disediakan oleh e-learning sudah sesuai?	Sesuai Cukup Sesuai Kurang Sesuai Tidak Sesuai	[ ] [ ] [ ] [ ]