MINISTRY OF DEVELOPMENT OF INFORMATION TECHNOLOGIES AND COMMUNICATIONS OF THE REPUBLIC OF UZBEKISTAN

TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES NAMED AFTER MUHAMMAD AL-KHWARIZMI NUKUS BRANCH

Department of Information technologies
Computer engineering (Computer engineering) direction

Permitted to defence
Head of the department
Aytmuratov B.Sh.
_____
«___»_____ 2017 y.

DEVELOPING THE VIRTUAL RECEPTION SYSTEM OF THE NUKUS BRANCH OF TUIT USING YII FRAMEWORK

GRADUATE QUALIFICATION WORK

Graduate: _____ Kadirbergenov N.S.

Supervisor: _____ Bisenbaev J.

Nukus 2017

Contents

INTRODUCTION

Nowadays commonly known as “virtual reception” system – is important to organize direct communication with people, to protect rights and liberties and legal benefits of physical and juridical people, to provide the activity of new and successful system in working with appeals of physical and juridical people.

Publicizing 2017th year “The year of dialogue with public and human interests” by the President shows that this system is very important.

Virtual reception system – gives the opportunity of sending their appeals from distance to juridical and physical people by using informational communication technologies. As mentioned in the Law “About physical and juridical people’s appeals” of the Republic of Uzbekistan, appeals can be in the kinds of applications, offers and complaints, moreover, appeals can be in oral, written and electron forms. Appeals, regardless of their kinds and forms, are in the same importance.

Application – appeal in which given the account of the requests about helping to put rights, liberties and benefits into action.

Offer – appeal which includes offers about developing state and society activity.

Complaint – appeal in which given the account of the demand in setting up destroyed rights and liberties and protecting legal benefits.

The qualification paper presents practical works about creating “Virtual reception” of the Nukus branch of TUIT by using Yii 2 PHP framework.

Yii is a free, open source web application development framework, written in PHP5, that promotes clean DRY design and encourages rapid development. It works to streamline your application development time and helps to ensure an extremely efficient, extensible, and maintainable end product. Being extremely performance-optimized, Yii is a perfect choice for any size project. However, it has been built with sophisticated, enterprise applications in mind. You have full control over the configuration from head-to-toe (presentation-to-persistence) to conform to

your enterprise development guidelines. It comes packaged with tools to help test and debug your application, and has clear and comprehensive documentation.

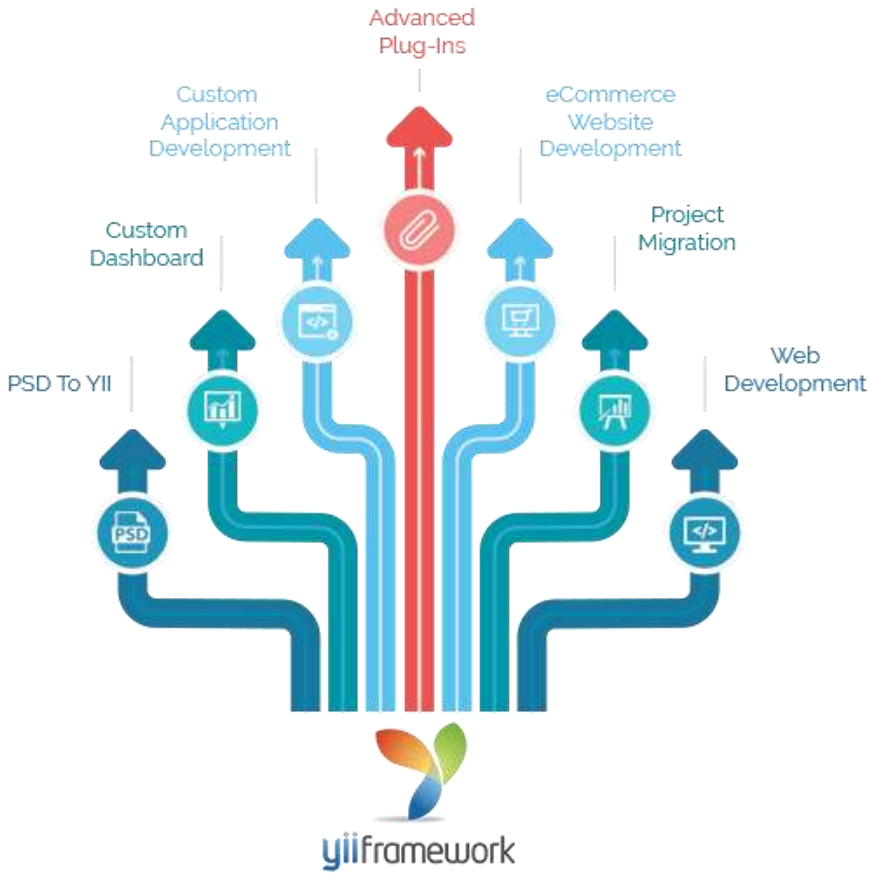
Like any good framework, Yii helps you create modern web applications quickly, and make sure they perform well. It pushes you to create secure and testable sites by doing a lot of the heavy lifting for you. You can easily use most of its features exactly as they are provided, or you can modify each one to suit your needs. I really encourage you to check it out for your next web project.

Consisting of introduction, five basic paragraphs conclusion, used literature and appendixes, the qualification paper presents opportunities, origin and steps of creating projects in Yii PHP framework.

§1. About Yii PHP framework

Yii is a free, open source web application development framework, written in PHP5, that promotes clean DRY design and encourages rapid development. It works to streamline your application development time and helps to ensure an extremely efficient, extensible, and maintainable end product. Being extremely performance-optimized, Yii is a perfect choice for any size project. However, it has been built with sophisticated, enterprise applications in mind. You have full control over the configuration from head-to-toe (presentation-to-persistence) to conform to your enterprise development guidelines. It comes packaged with tools to help test and debug your application, and has clear and comprehensive documentation.

Yii helps Web developers build complex applications and deliver them on-time. Yii is pronounced as Yee or [ji:], and is an acronym for **"Yes It Is!"** This is often the accurate, and most concise response to inquires from those new to Y ii: Is it fast? ... Is it secure? ... Is it professional? ... Is it right for my next project? ... Yes, it is!



To learn more about what Yii brings to the table, check out the features section.

History. Yii is the brainchild of its founder, Qiang Xue, who started the Yii project on January 1, 2008. Qiang previously developed and maintained the Prado framework. The years of experience gained and developer feedback gathered from that project solidified the need for an extremely fast, secure and professional framework that is tailor-made to meet the expectations of Web 2.0 application development. On December 3, 2008, after nearly one year's development, Yii 1.0 was formally released to the public.

Its extremely impressive performance metrics when compared to other PHP-based frameworks immediately drew very positive attention and its popularity and adoption continues to grow at an ever increasing rate.

On October 2014 Yii 2.0.0 was released which is a complete rewrite over the previous version that was made in order to build a state-of-the-art PHP framework

by keeping the original simplicity and extensibility of Yii while adopting the latest technologies and features to make it even better.

For a high performance framework, Yii's breadth of capabilities and feature set is quite extraordinary, and it's one of the reasons I was attracted to the framework. You can see and try some examples of Yii code in action at the Yii Playground.

Here are some highlights:

- Model-View-Controller architecture. Just like Ruby on Rails, now you can leverage MVC in your PHP apps.
- Database Access Objects (DAO), Active Record, and programmatic Database migrations simplify the challenges of building database-powered web applications.
- Form input, validation, and Ajax support is built-in. Yii makes forms pretty easy.
- Built-in authentication and powerful user management extensions make launching new web applications easy.
- Yii's built-in code generation tool, Gíi, speeds your app's development scaffolding in MVC fashion
- Console Yii. You can run Yii from the command line or as a daemon. With this, it's possible to build high performance background tasks in PHP.
- Theming options such as the Bootstrap extension makes building great looking responsive apps much simpler
- Layered caching support Yii makes it easy to implement the kinds of caching that make sense to your application
- Security. Yii greatly minimizes the typical risk factors of running services with PHP and MySQL
- Integration with other frameworks. It's easy to use Zend or PEAR features in Yii
- Extensions. Yii's community offers a variety of free, open source plugins and widgets

- Internationalization. Yii supports I18N and makes it easy to provide localized versions of your app.
- Error handling, logging and testing - yes, Yii delivers.

The Yii community also purchased an unlimited license to the beautiful web-based rich text editor, Redactor - so you can use it any of your own Yii apps. This is great for rich formatted input or CMS features.

Yii is the brainchild of its founder, Qiang Xue, who started the Yii project on January 1, 2008. Qiang previously developed and maintained the Prado framework. The years of experience gained and developer feedback gathered from that project solidified the need for an extremely fast, secure and professional framework that is tailor-made to meet the expectations of Web 2.0 application development. On December 3, 2008, after nearly one year's development, Yii 1.0 was formally released to the public.

It’s extremely impressive performance metrics when compared to other PHP-based frameworks immediately drew very positive attention and its popularity and adoption continues to grow at an ever increasing rate.

On October 2014 Yii 2.0.0 was released which is a complete rewrite over the previous version that was made in order to build a state-of-the-art PHP framework by keeping the original simplicity and extensibility of Yii while adopting the latest technologies and features to make it even better.

What's New in Release 2.0?

Yii 2.0 is poised for release (follow the development roadmap). The Yii2 beta notes best summarize the improvements from Yii 1.x.

In part because of how advanced Yii 1.x was, there isn't a single amazing feature to Yii2, but there are a broad set of substantial improvements which will make life increasingly easier for Yii developers.

Here are a few highlights:

- Support for PSR-4 class autoloading, simpler namespaces, faster loading and improved usability for developers

- Performance and security improvements
- RESTful API framework integration to make building APIs easier
- Codeception testing integration
- Database and active record improvements including batched queries, support for sub-queries and inverse relations
- Improved URL handling and processing

§2. 7 reasons to choose the Yii 2 framework.

1. Easy to Install.

For web developers, time is money, and no one wants to spend their precious time on a complicated installation and configuration process.

Installation is handled using Composer. If you want a description of the installation process, Sitepoint recently published a great article here. I tend to favor using the basic application template, even when my site has a separate front- and backend component. Instead, I opt to use a Module for the backend portion of my sites. (Yii Modules are best described as mini-applications which reside inside your main application).

Note: Many of the directory references in later examples use the directory structure from the simple template.

2. Utilizes modern technologies.

Yii is a pure OOP framework, and takes advantage of some of PHP's more advanced features, including late static binding, SPL classes and interfaces, and anonymous functions.

All classes are namespaced, which allows you to take advantage of their PSR-4 compliant autoloader. That means that including Yii's HTML helper class is as simple as:

```
use yii\helpers\Html;
```

Yii also allows you to define aliases to help simplify your namespaces. In the above example, that `use` statement will load a class definition which is located by default in the directory **`/vendor/yiisoft/yii2/helpers`**.

```
public static $aliases = ['@yii' => __DIR__];
```

The framework itself is installed using Composer, as are its extensions. Even the process of publishing extensions is as easy as creating your own

`composer.json`, hosting your code at Github, and listing your extension on Packagist.

3. Highly extensible.

Yii is like a suit that looks great off of the rack, but is also very easy to tailor to fit your needs. Virtually every component of the framework is extensible. A simple example is the addition of a unique body ID to your views. (In case you're interested in knowing why you might want to do this, take a look at this article).

First, I would create a file in my `app\components` directory with the name `View.php`, and add the following:

```
namespace app\components;

class View extends yii\web\View {

    public $bodyId;

    /* Yii allows you to add magic getter methods by prefacing method names with "get" */

    public function getBodyIdAttribute() {
        return ($this->bodyId != '') ? 'id="' . $this->bodyId . '"' : '';
    }

}
```

Then, in my main layout file (`app\views\layouts\main.php`), I would add the following to the body tag of my HTML:

```
<body <?=$this->BodyIdAttribute?>>
```

And finally, I would add the following to my main configuration file to let Yii know to use my extended `View` class instead of its own default:

```
return [
    // ...
    'components' => [
        // ...
        'view' => [
            'class' => 'app\components\View'
        ]
    ]
];
```

4. Encourages testing.

Yii is tightly integrated with Codeception. Codeception is a great PHP testing framework that helps simplify the process of creating unit, functional and acceptance tests for your application. Because you ARE writing automated tests for all your applications, right?

The Codeception extension makes it simple to configure your application during testing. Simply edit the provided `/tests/_config.php` file to configure your test application. For example:

```
return [
    'components' => [
        'mail' => [
            'useFileTransport' => true,
        ],
        'urlManager' => [
            'showScriptName' => true,
        ],
        'db' => [
                'dsn' => 'mysql:host=localhost;dbname=mysqldb_test',
        ],
    ],
];
```

Using this configuration, the following would happen:

1. Any emails sent during your functional and acceptance tests would be written to a file instead of being sent.
2. The URLs in your tests would take on the format `index.php/controller/action` rather than `/controller/ action`
3. Your tests would use your test database, rather than your production database.

A special module for the Yii Framework also exists within Codeception. It adds several methods to the `TestGuy` class, which help you work with Active Record (Yii's ORM) during functional tests. For instance, if you wanted to see if your registration form successfully created a new `User` with the username "testuser", you could do the following:

```
$I->amOnPage('register');
$I->fillField('username', 'testuser');
$I->fillField('password', 'qwerty');
$I->click('Register');
$I->seeRecord('app\models\User', array('name' => 'testuser'));
```

5. Simplifies security.

Security is a crucial part of any web application, and fortunately Yii has some great features to help ease your mind.

Yii comes with a Security application component that exposes several methods to help assist in creating a more secure application. Some of the more useful methods are:

- generatePasswordHash: Generates a secure hash from a password and a random salt. This method makes a random salt for you, and then creates a hash from the supplied string using PHP's crypt function.
- validatePassword: This is the companion function to `generatePasswordHash`, and allows you to check whether the user supplied password matches your stored hash.
- generateRandomKey: Allows you to create a random string of any length

Yii automatically checks for a valid CSRF token on all unsafe HTTP request methods (PUT, POST, DELETE), and will generate and output a token when you use the ActiveForm::begin() method to create your opening form tag. This feature can be disabled by editing your main configuration file to include the following:

```
return [
        'components' => [
            'request' => [
                'enableCsrfValidation' => false,
            ]
    ];
```

In order to protect against XSS, Yii provides another helper class called HtmlPurifier. This class has a single static method named process, and will filter your output using the popular filter library of the same name.

Yii also includes ready-to-use classes for user authentication and authorization. Authorization is broken into two types: ACF (Access Control Filters) and RBAC (Role-Based Access Control).

The simpler of the two is ACF, and is implemented by adding the following to the behaviors method of your controller:

```
use yii\filters\AccessControl;

class DefaultController extends Controller {
    // ...
    public function behaviors() {
        return [
            // ...
            'class' => AccessControl::className(),
            'only' => ['create', 'login', 'view'],
                'rules' => [
                [
                    'allow' => true,
                    'actions' => ['login', 'view'],
                    'roles' => ['?']
                ],
                [
                    'allow' => true,
                    'actions' => ['create'],
                    'roles' => ['@']
                ]
            ]
        ];
    }
    // ...
}
```

The preceding code tells `DefaultController`to allow guest users to access the `login` and `view` actions, but not the `create` action. (`?` is an alias for anonymous users, and `@` refers to authenticated users).

RBAC is a more powerful method of specifying which users can perform specific actions throughout your application. It involves creating roles for your users, defining permissions for your app, and then enabling those permissions for their intended roles. You could use this method if you wanted to create a `Moderator` role, and allow all users assigned to this role to approve articles.

You can also define rules using RBAC, which allow you, under specific conditions, to grant access to certain aspects of your application. For instance, you

could create a rule that allows users to edit their own articles, but not those created by others.

6. Shorten development time.

Most projects involve a certain amount of repetitive tasks that no one wants to waste time with. Yii gives us a few tools to help you spend less time on those tasks, and more time customizing your application to suit your clients' needs.

One of the most powerful of these tools is called "Gíi". Gíi is a web-based code scaffolding tool, which allows you to quickly create code templates for:

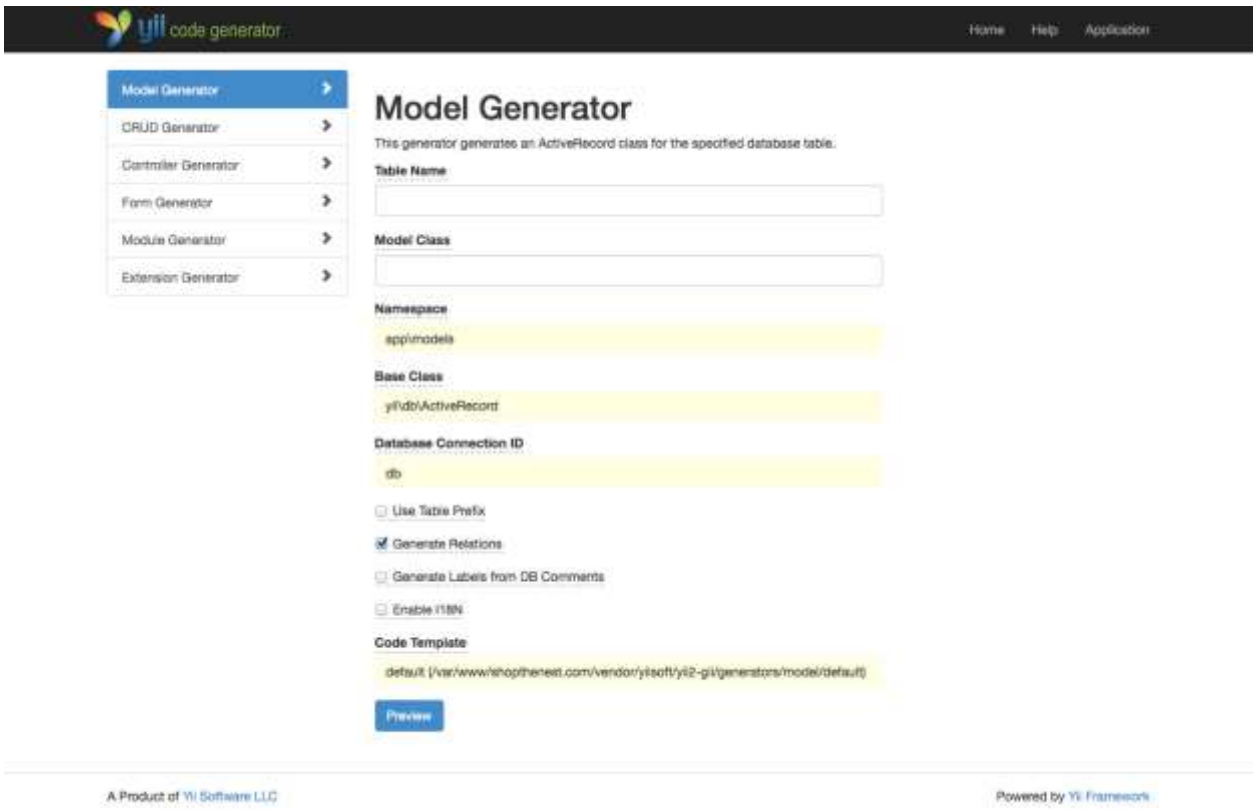
- **Models**
- **Controllers**
- **Forms**
- **Modules**
- **Extensions**
- **CRUD controller actions and views**

Gíi is highly configurable. You can set it to only load in certain environments. Simply edit your web configuration file as follows:

```
if (YII_ENV_DEV) {
    // ...
    $config['modules']['gii'] = [
        'class' => 'yii\gii\Module',
        'allowedIPs' => ['127.0.0.1', '::1']
    ]
}
```

This ensures that Gíi will only load when the Yii environment variable is set to *development*, and that it will only load when accessed via localhost.

Now let's take a look at the model generator:



The table name uses a typeahead widget to try to guess which table your model is associated with, and all fields have a rollover tooltip to remind you how to fill them out. You can preview code before you ask Gíi to generate it, and all the code templates are completely customizable.

There are also several command-line tools available to help create code templates for database migrations, message translations (I18N) and database fixtures for your automated tests. For instance, you can create a new database migration file with this command:

```
yii migrate/create create_user_table
```

This will create a new migration template in {appdir}/migrations that looks something like this:

```php
<?php

    use yii\db\Schema;

    class m140924_153425_create_user_table extends \yii\db\Migration
    {
        public function up()
        {

        }

        public function down()
        {
            echo "m140924_153425_create_user_table cannot be reverted.\n";

            return false;
        }
}
```

So let's say I wanted to add a few columns to this table. I would simply add the following to the `up` method:

`public function up()`

```php
public function up()
{
    $this->createTable('user', [
        'id' => Schema::TYPE_PK,
        'username' => Schema::TYPE_STRING . ' NOT NULL',
        'password_hash' => Schema:: TYPE_STRING . ' NOT NULL'
    ], null);
}
```

And then to make sure I can reverse the migration, I would edit the `down` method:

```php
public function down()
{
    $this->dropTable('user');
}
```

Creating the table would simply involve running a command on the command line:

```
./yii migrate
```

and to remove the table:

```
./yii migrate/down
```

7. Easy to tune for better performance.

Everybody knows that a slow website creates disgruntled users, so Yii provides you with several tools to help you squeeze more speed out of your application.

All Yií's cache components extend from yii/caching/Cache, which allows you to choose whichever caching system you want while using a common API. You can even register multiple cache components simultaneously. Yii currently supports database and file system caching, as well as APC, Memcache, Redis, WinCache, XCache and Zend Data Cache.

By default, if you're using Active Record then Yii runs an extra query to determine the schema of the table(s) involved in generating your model. You can set the application to cache these schema by editing your main configuration file as follows:

```
return [
    // ...
    'components' => [
        // ...
        'db' => [
            // ...
            'enableSchemaCache' => true,
            'schemaCacheDuration' => 3600,
            'schemaCache' => 'cache',
        ],
        'cache' => [
            'class' => 'yii\caching\FileCache',
        ],
    ],
];
```

Finally, Yii has a command line tool to facilitate the minification of frontend assets. Simply run the following command to generate a configuration template:

```
./yii asset/template config.php
```

Then edit the configuration to specify which tools you want to you perform your minification (e.g. Closure Compiler, YUI Compressor, or UglifyJS). The generated configuration template will look like this:

```
<?php
    return [
        'jsCompressor' => 'java -jar compiler.jar --js {from} --js_output_file {to}',
        'cssCompressor' => 'java -jar yuicompressor.jar --type css {from} -o {to}',
        'bundles' => [
            // 'yii\web\YiiAsset',
            // 'yii\web\JqueryAsset',
        ],
        'targets' => [
            'app\config\AllAsset' => [
                'basePath' => 'path/to/web',
                'baseUrl' => '',
                'js' => 'js/all-{hash}.js',
                'css' => 'css/all-{hash}.css',
            ],
        ],
        'assetManager' => [
            'basePath' => __DIR__,
            'baseUrl' => '',
        ],
    ];
```

Next, run this console command in order to perform the compression.

```
yii asset config.php /app/assets_compressed.php
```

And finally, edit your web application configuration file to use the compressed assets.

```
'components' => [
    // ...
    'assetManager' => [
        'bundles' => require '/app/assets_compressed.php'
    ]
]
```

Note: You will have to download and install these external tools manually.

§3. How to program with Yii2: getting started.

This tutorial will walk you through installing Yii2, setting up your local development environment, building a simple Hello World application, setting up your remote production environment for hosting and deploying your code from a *GitHub* repository.

Installing the Yii2

1. Installing the Composer.

Yii2 requires Composer, a popular dependency manager for PHP. If you don't already have Composer installed, do the following:

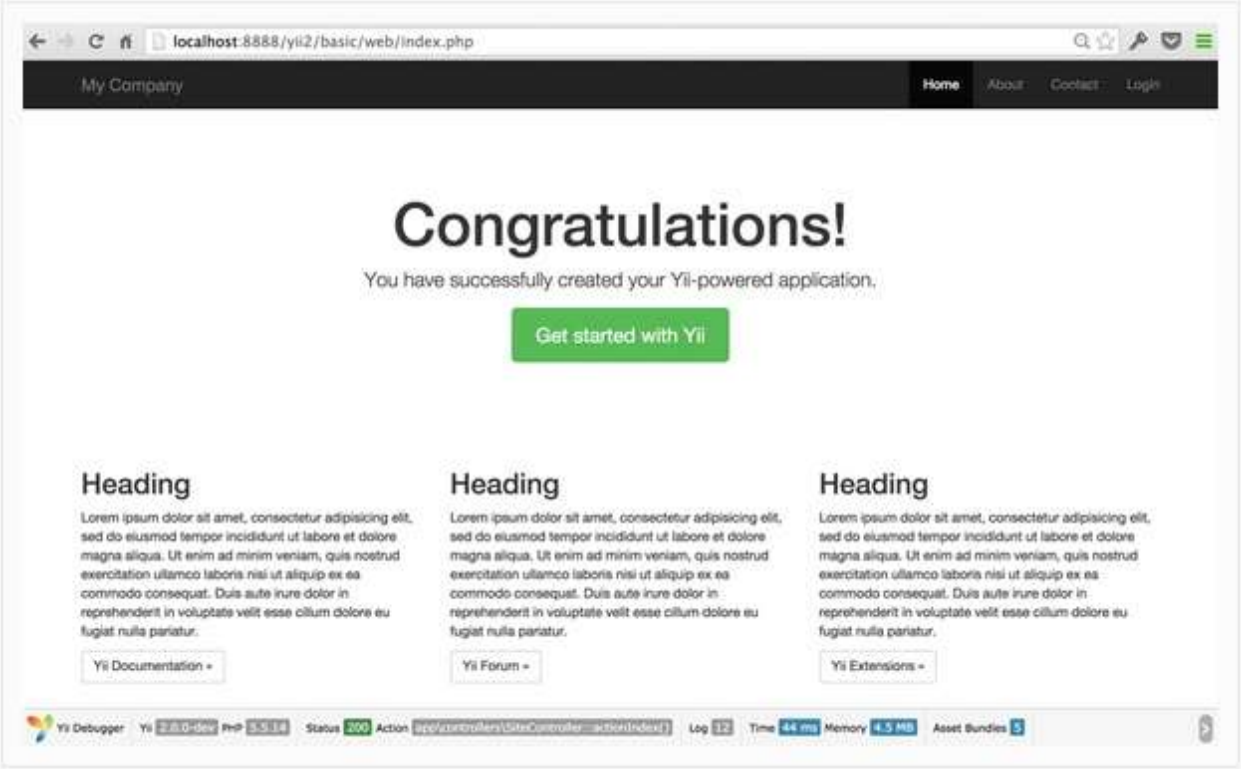
```
curl -s http://getcomposer.org/installer | php
mv composer.phar /usr/local/bin/composer
```

2. Installing Yii2.

Then, use Composer to install Yii2. The installation request requires that you use your own Github account credentials; sign up if you don't have an account.
Let's call this first project, "hello":

```
cd ~/Sites
composer global require "fxp/composer-asset-plugin:1.0.0-beta2"
composer create-project --prefer-dist yiisoft/yii2-app-basic hello
```

Yii2 provides two different installation templates depending on the type of application you're developing: **basic** and **advanced**. For this tutorial, we'll use the basic application which is installed by default. The advanced application template provides a **front-end**, **back end** and console access points for a more advanced web application, like a **WordPress** blog, it's administrative dashboard and background cron tasks.



3. Yii2 application structure.

We are assuming you have installed the basic yii2 application template.

Yii2 follows the MVC(Model View Controller) pattern.

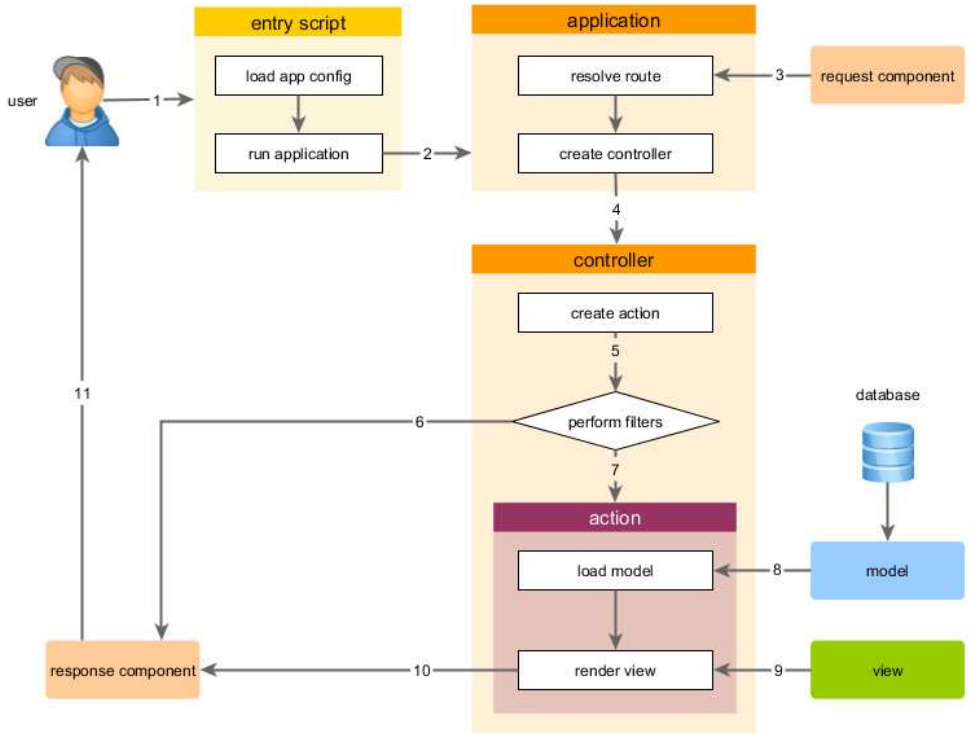
The pattern just states that we should keep the presentation of data separate from the methods that interact with the data. That means if you have db query or a business logic it should be in a model. Your html (presentation) should be in a view and the controller is the link between the two. Yes it is that simple. Now lets take a look of what we have in the yii2 project we created.

1. **basic** Application base path (this is the name you gave when creating the project using composer)
2. **composer.json/** Remember we used Composer to install the application this file is generated by it to keep which packages are installed and there dependencies.
3. **config** contains application and other configurations for example –
   - **console.php/** The console application configuration

- **db.php/** Will have the details of the your connection information to your database like the username and password
- **params.php/** You can declare the params you need in your application these can be accessed from anywhere. by using Yii2::$app->params[‘value’]
- **web.php/** Contains the Web application configuration like

4. **commands/** contains console command classes
5. **controllers/** contains controller classes
6. **models/** contains model classes
7. **runtime/** contains files generated by Yii2 during runtime, such as logs and cache files
8. **vendor/** contains the installed Composer packages, including the Yii2 framework itself
9. **views/** contains view files
10. **web/** application Web root, contains Web accessible files
    - **assets** contains published asset files (javascript and css) by Yii2
    - **css** your css files should go in this folder
    - **index.php/** the entry (or bootstrap) script for the application.

Next lets look at the request lifecycle



1. A user makes a request to the *web/index.php* web directory should be the only accessible folder through http requests
2. The *index.php* loads the application configurations and creates an application instance to handle the request. Checkout the *web/index.php* now
3. The application resolves the requested route with the help of the request application component.
4. The application creates a controller object to handle the request.
5. The controller creates an action instance and performs the filters for the action.
6. If any filter fails, the action is cancelled.
7. If all filters pass, the action is executed.
8. The action loads a data model, possibly from a database.
9. The action renders a view, providing it with the data model.
10. The rendered result is returned to the response application component.
11. The response component sends the rendered result to the user's browser.

Saying hello.

This section describes how to create a new "Hello" page in your application. To achieve this goal, you will create an action and a view:

- The application will dispatch the page request to the action
- and the action will in turn render the view that shows the word "Hello" to the end user.

Through this tutorial, you will learn three things:

1. how to create an action to respond to requests,
2. how to create a view to compose the response's content, and
3. how an application dispatches requests to actions.

Creating an action.

For the "Hello" task, you will create a say action that reads a message parameter from the request and displays that message back to the user. If the request does not provide a message parameter, the action will display the default "Hello" message.

Actions must be declared in controllers. For simplicity, you may declare the say action in the existing SiteController. This controller is defined in the class file controllers/SiteController.php. Here is the start of the new action:

```
<?php

namespace app\controllers;

use yii\web\Controller;

class SiteController extends Controller
{
    // ...existing code...

    public function actionSay($message = 'Hello')
    {
        return $this->render('say', ['message' => $message]);
    }
}
```

Creating a View.

Views are scripts you write to generate a response's content. For the "Hello" task, you will create a say view that prints the message parameter received from the action method:

```
<?php
use yii\helpers\Html;
?>
<?= Html::encode($message) ?>
```

Trying it out.

After creating the action and the view, you may access the new page by accessing the following URL:

```
http://hostname/index.php?r=site%2Fsay&message=Hello+World
```

Hello World

Working with forms.

This section describes how to create a new page with a form for getting data from users. The page will display a form with a name input field and an email input field. After getting those two pieces of information from the user, the page will echo the entered values back for confirmation.

To achieve this goal, besides creating an action and two views, you will also create a model.

Through this tutorial, you will learn how to:

- create a model to represent the data entered by a user through a form,
- declare rules to validate the data entered,
- build an HTML form in a view.

Creating a Model.

The data to be requested from the user will be represented by an EntryForm model class as shown below and saved in the file models/EntryForm.php. Please refer to the Class Autoloading section for more details about the class file naming convention.

```php
<?php

namespace app\models;

use Yii;
use yii\base\Model;

class EntryForm extends Model
{
    public $name;
    public $email;

    public function rules()
    {
        return [
            [['name', 'email'], 'required'],
            ['email', 'email'],
        ];
    }
}
```

Creating an Action.

Next, you'll need to create an entry action in the site controller that will use the new model. The process of creating and using actions was explained in the Saying Hello section.

```php
<?php

namespace app\controllers;

use Yii;
use yii\web\Controller;
use app\models\EntryForm;

class SiteController extends Controller
{
    // ...existing code...

    public function actionEntry()
    {
        $model = new EntryForm();

        if ($model->load(Yii::$app->request->post()) && $model->validate()) {
            // valid data received in $model

            // do something meaningful here about $model ...

            return $this->render('entry-confirm', ['model' => $model]);
        } else {
            // either the page is initially displayed or there is some validation error
            return $this->render('entry', ['model' => $model]);
        }
    }
}
```

Creating Views.

Finally, create two view files named entry-confirm and entry. These will be rendered by the entry action, as just described.

The entry-confirm view simply displays the name and email data. It should be stored in the file *views/site/entry-confirm.php*.

```
<?php
use yii\helpers\Html;
?>
<p>You have entered the following information:</p>

<ul>
    <li><label>Name</label>: <?= Html::encode($model->name) ?></li>
    <li><label>Email</label>: <?= Html::encode($model->email) ?></li>
</ul>
```

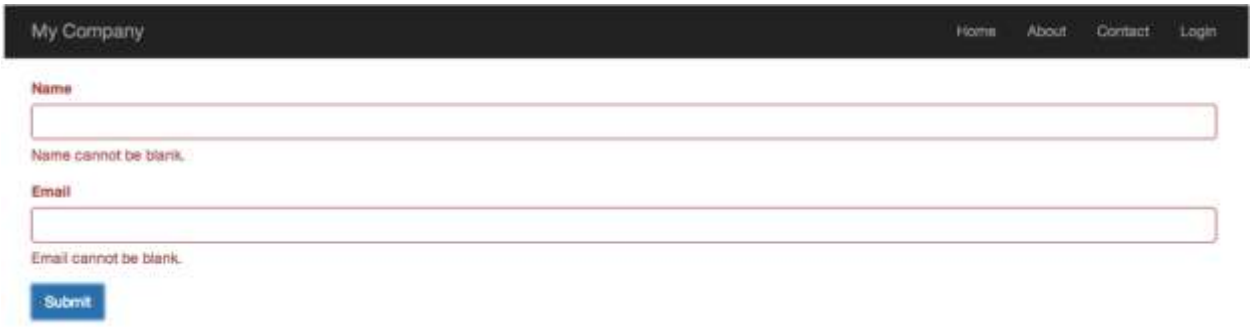
The entry view displays an HTML form. It should be stored in the file *views/site/entry.php*.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<?php $form = ActiveForm::begin(); ?>

    <?= $form->field($model, 'name') ?>

    <?= $form->field($model, 'email') ?>

    <div class="form-group">
        <?= Html::submitButton('Submit', ['class' => 'btn btn-primary']) ?>
    </div>

<?php ActiveForm::end(); ?>
```

Trying it out.

To see how it works, use your browser to access the following URL:

```
http://hostname/index.php?r=site%2Fentry
```



Security.

Good security is vital to the health and success of any application. Unfortunately, many developers cut corners when it comes to security, either due to a lack of understanding or because implementation is too much of a hurdle. To make your Yii2 powered application as secure as possible, Yii2 has included several excellent and easy to use security features.

- Authentication
- Authorization
- Working with Passwords
- Cryptography
- Views security
- Auth Clients.

Authentication

Authentication is the process of verifying the identity of a user. It usually uses an identifier (e.g. a username or an email address) and a secret token (e.g. a password or an access token) to judge if the user is the one whom he claims as. Authentication is the basis of the login feature.

Yii2 provides an authentication framework which wires up various components to support login. To use this framework, you mainly need to do the following work:

- Configure the user application component;
- Create a class that implements the yii2\web\IdentityInterface interface.

Configuring yii2\web\User.

The user application component manages the user authentication status. It requires you to specify an identity class which contains the actual authentication logic. In the following application configuration, the identity class for user is configured to be app\models\User whose implementation is explained in the next subsection:

```
return [
    'components' => [
        'user' => [
            'identityClass' => 'app\models\User',
        ],
    ],
];
```

You mainly use yii2\web\User in terms of the user application component. You can detect the identity of the current user using the expression `Yii2::$app->user->identity`. It returns an instance of the identity class representing the currently logged-in user, or null if the current user is not authenticated (meaning a guest). The following code shows how to retrieve other authentication-related information from `yii2\web\User:`

```
// the current user identity. `null` if the user is not authenticated.
$identity = Yii::$app->user->identity;

// the ID of the current user. `null` if the user not authenticated.
$id = Yii::$app->user->id;

// whether the current user is a guest (not authenticated)
$isGuest = Yii::$app->user->isGuest;
```

To login a user, you may use the following code:

```
// find a user identity with the specified username.
// note that you may want to check the password if needed
$identity = User::findOne(['username' => $username]);

// logs in the user
Yii::$app->user->login($identity);
```

The `yii2\web\User::login()` method sets the identity of the current user to the yii2\web\User. If session is enabled, it will keep the identity in the session so that the user authentication status is maintained throughout the whole session. If cookie-based login (i.e. "remember me" login) is enabled, it will also save the identity in a cookie so that the user authentication status can be recovered from the cookie as long as the cookie remains valid.

In order to enable cookie-based login, you need to configure yii2\web\User::$enableAutoLogin to be true in the application configuration. You also need to provide a duration time parameter when calling the yii2\web\User::login() method.

To logout a user, simply call

```
Yii::$app->user->logout();
```

Authorization

Authorization is the process of verifying that a user has enough permission to do something. Yii2 provides two authorization methods: Access Control Filter (ACF) and Role-Based Access Control (RBAC).

Access Control Filter

Access Control Filter (ACF) is a simple authorization method implemented as yii2\filters\AccessControl which is best used by applications that only need some simple access control. As its name indicates, ACF is an action filter that can be used in a controller or a module. While a user is requesting to execute an action, ACF will check a list of access rules to determine if the user is allowed to access the requested action.

The code below shows how to use ACF in the site controller:

```
use yii2\web\Controller;
use yii2\filters\AccessControl;

class SiteController extends Controller
{
    public function behaviors()
    {
        return [
            'access' => [
                'class' => AccessControl::className(),
                'only' => ['login', 'logout', 'signup'],
                'rules' => [
                    [
                        'allow' => true,
                        'actions' => ['login', 'signup'],
                        'roles' => ['?'],
                    ],
                    [
                        'allow' => true,
                        'actions' => ['logout'],
                        'roles' => ['@'],
                    ],
                ],
            ],
        ];
    }
    // ...
}
```

In the code above ACF is attached to the site controller as a behavior. This is the typical way of using an action filter. The only option specifies that the ACF should only be applied to the login, logout and signup actions. All other actions in the site controller are not subject to the access control. The rules option lists the access rules, which reads as follows:

Allow all guest (not yet authenticated) users to access the login and signup actions. The roles option contains a question mark ? which is a special token representing "guest users".

Allow authenticated users to access the logout action. The @ character is another special token representing "authenticated users".

ACF performs the authorization check by examining the access rules one by one from top to bottom until it finds a rule that matches the current execution context. The allow value of the matching rule will then be used to judge if the user is authorized or not. If none of the rules matches, it means the user is NOT authorized, and ACF will stop further action execution.

When ACF determines a user is not authorized to access the current action, it takes the following measure by default:

You may customize this behavior by configuring the yii2\filters\AccessControl::$denyCallback property like the following:

```
[
    'class' => AccessControl::className(),
    ...
    'denyCallback' => function ($rule, $action) {
        throw new \Exception('You are not allowed to access this page');
    }
]
```

Role Based Access Control (RBAC)

Role-Based Access Control (RBAC) provides a simple yet powerful centralized access control. Please refer to the Wikipedia for details about comparing RBAC with other more traditional access control schemes.

Yii2 implements a General Hierarchical RBAC, following the NIST RBAC model. It provides the RBAC functionality through the authManager application component. Using RBAC involves two parts of work. The first part is

to build up the RBAC authorization data, and the second part is to use the authorization data to perform access check in places where it is needed. To facilitate our description next, we will first introduce some basic RBAC concepts.

Basic Concepts. A role represents a collection of permissions (e.g. creating posts, updating posts). A role may be assigned to one or multiple users. To check if a user has a specified permission, we may check if the user is assigned with a role that contains that permission.

Associated with each role or permission, there may be a rule. A rule represents a piece of code that will be executed during access check to determine if the corresponding role or permission applies to the current user. For example, the "update post" permission may have a rule that checks if the current user is the post creator. During access checking, if the user is NOT the post creator, he/she will be considered not having the "update post" permission.

Both roles and permissions can be organized in a hierarchy. In particular, a role may consist of other roles or permissions; and a permission may consist of other permissions. Yii2 implements a partial order hierarchy which includes the more special tree hierarchy. While a role can contain a permission, it is not true vice versa.

Configuring RBAC. Before we set off to define authorization data and perform access checking, we need to configure the authManager application component. Yii2 provides two types of authorization managers: yii2\rbac\PhpManager and yii2\rbac\DbManager. The former uses a PHP script file to store authorization data, while the latter stores authorization data in a database. You may consider using the former if your application does not require very dynamic role and permission management.

Using PhpManager

The following code shows how to configure the authManager in the application configuration using the yii2\rbac\PhpManager class:

```
return [
    // ...
    'components' => [
        'authManager' => [
```

```
            'class' => 'yii2\rbac\PhpManager',
        ],
        // ...
    ],
];
```

Using DbManager

The following code shows how to configure the authManager in the application configuration using the yii2\rbac\DbManager class:

```
return [
    // ...
    'components' => [
        'authManager' => [
            'class' => 'yii2\rbac\DbManager',
        ],
        // ...
    ],
];
```

DbManager uses four database tables to store its data:

- itemTable: the table for storing authorization items. Defaults to "auth_item".
- itemChildTable: the table for storing authorization item hierarchy. Defaults to "auth_item_child".
- assignmentTable: the table for storing authorization item assignments. Defaults to "auth_assignment".
- ruleTable: the table for storing rules. Defaults to "auth_rule".

Before you can go on you need to create those tables in the database. To do this, you can use the migration stored in @yii2/rbac/migrations:

```
yii2 migrate --migrationPath=@yii2/rbac/migrations
```

Read more about working with migrations from different namespaces in Separated Migrations section. The authManager can now be accessed via \Yii2::$app->authManager.

Building Authorization Data

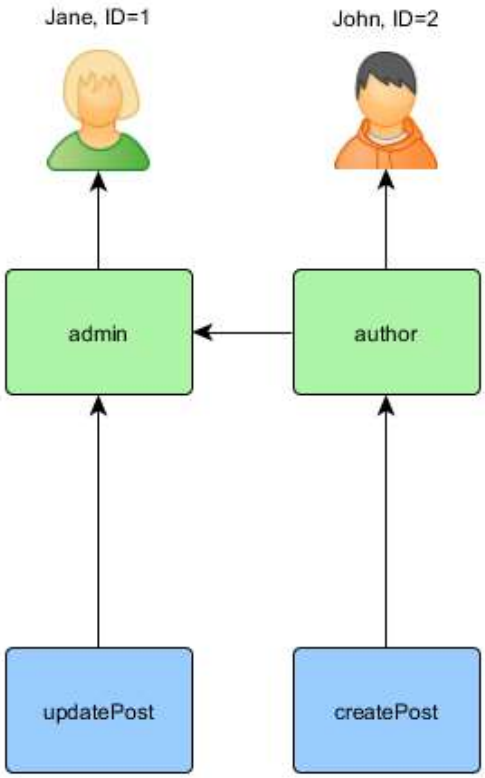
Building authorization data is all about the following tasks:

- defining roles and permissions;
- establishing relations among roles and permissions;
- defining rules;
- associating rules with roles and permissions;
- assigning roles to users.

Depending on authorization flexibility requirements the tasks above could be done in different ways. If your permissions hierarchy doesn't change at all and you have a fixed number of users you can create a console command that will initialize authorization data once via APIs offered by authManager:

```
<?php
namespace app\commands;

use Yii2;
use yii2\console\Controller;

class RbacController extends Controller
{
    public function actionInit()
    {
        $auth = Yii2::$app->authManager;

        // add "createPost" permission
        $createPost = $auth->createPermission('createPost');
        $createPost->description = 'Create a post';
        $auth->add($createPost);

        // add "updatePost" permission
        $updatePost = $auth->createPermission('updatePost');
        $updatePost->description = 'Update post';
        $auth->add($updatePost);

        // add "author" role and give this role the "createPost" permission
        $author = $auth->createRole('author');
        $auth->add($author);
        $auth->addChild($author, $createPost);

        // add "admin" role and give this role the "updatePost" permission
        // as well as the permissions of the "author" role
        $admin = $auth->createRole('admin');
        $auth->add($admin);
        $auth->addChild($admin, $updatePost);
        $auth->addChild($admin, $author);

        // Assign roles to users. 1 and 2 are IDs returned by
IdentityInterface::getId()
        // usually implemented in your User model.
        $auth->assign($author, 2);
        $auth->assign($admin, 1);
    }
}
```

After executing the command with yii2 rbac/init we'll get the following hierarchy:

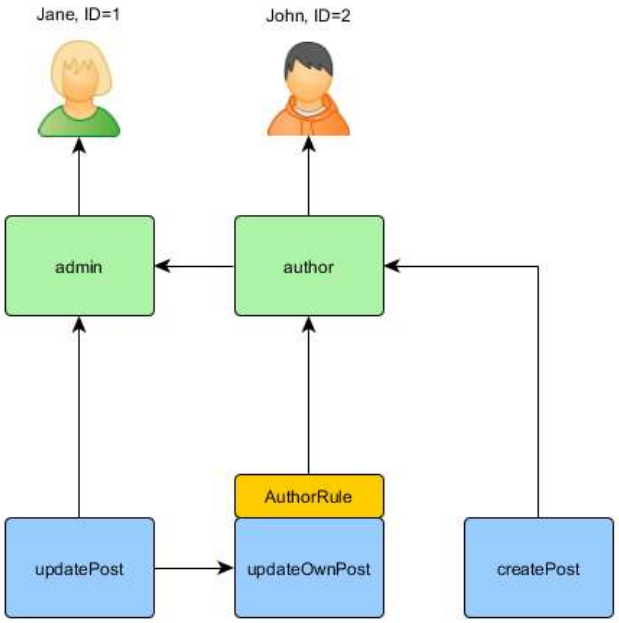


Author can create post, admin can update post and do everything author can. If your application allows user signup you need to assign roles to these new users once. For example, in order for all signed up users to become authors in your advanced project template you need to modify `frontend\models\SignupForm::signup()` as follows:

```
public function signup()
{
    if ($this->validate()) {
        $user = new User();
        $user->username = $this->username;
        $user->email = $this->email;
        $user->setPassword($this->password);
        $user->generateAuthKey();
        $user->save(false);

        // the following three lines were added:
        $auth = \Yii2::$app->authManager;
        $authorRole = $auth->getRole('author');
        $auth->assign($authorRole, $user->getId());

        return $user;
    }

    return null;
}
```

Using Rules

As aforementioned, rules add additional constraint to roles and permissions. A rule is a class extending from yii2\rbac\Rule. It must implement the execute() method. In the hierarchy we've created previously author cannot edit his own post. Now we have got the following hierarchy:

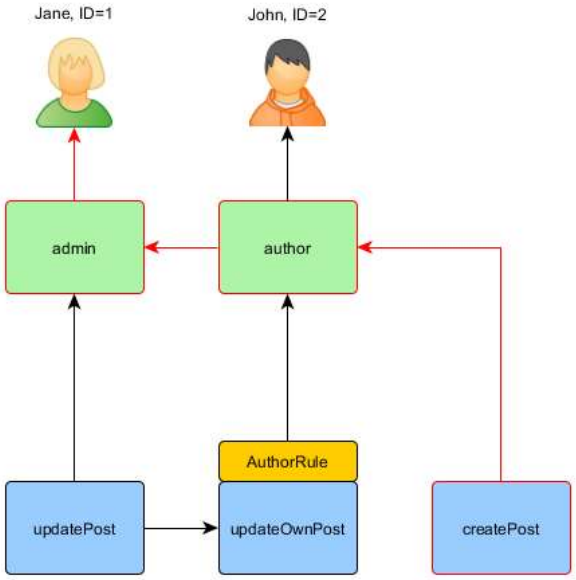


Access Check

With the authorization data ready, access check is as simple as a call to the `yii2\rbac\ManagerInterface::checkAccess()` method. Because most access check is about the current user, for convenience Yii2 provides a shortcut method yii2\web\User::can(), which can be used like the following:

```
if (\Yii2::$app->user->can('createPost')) {
    // create post
}
```

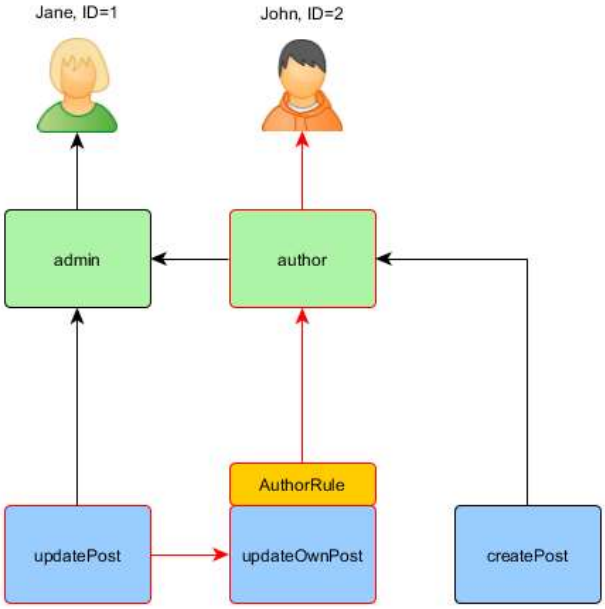
If the current user is Jane with *ID=1* we are starting at *createPost* and trying to get to Jane:



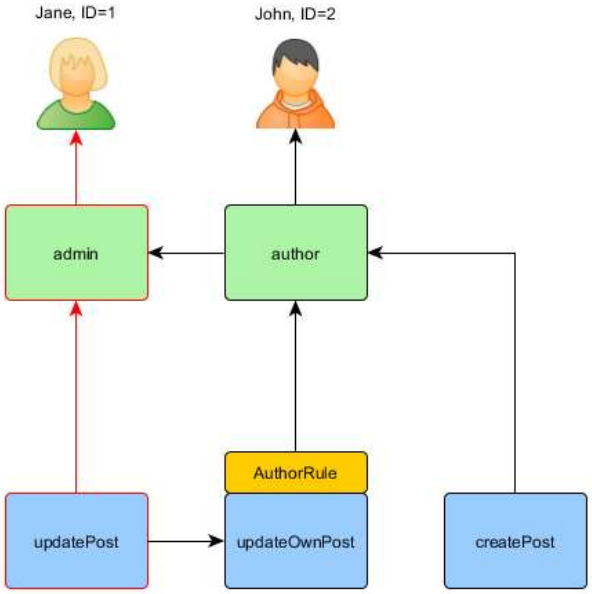
In order to check if a user can update a post, we need to pass an extra parameter that is required by *AuthorRule* described before:

```
if (\Yii2::$app->user->can('updatePost', ['post' => $post])) {
    // update post
}
```

Here is what happens if the current user is John:



We are starting with the updatePost and going through updateOwnPost. In order to pass the access check, AuthorRule should return true from its execute() method. The method receives its $params from the can() method call so the value is ['post' => $post]. If everything is fine, we will get to author which is assigned to John. In case of Jane it is a bit simpler since she is an admin:



Inside your controller there are a few ways to implement authorization. If you want granular permissions that separate access to adding and deleting, then you need to check access for each action. You can either use the above condition in each action method, or use *yii2\filters\AccessControl:*

```
public function behaviors()
{
    return [
        'access' => [
            'class' => AccessControl::className(),
            'rules' => [
                [
                    'allow' => true,
                    'actions' => ['index'],
                    'roles' => ['managePost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['view'],
                    'roles' => ['viewPost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['create'],
                    'roles' => ['createPost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['update'],
                    'roles' => ['updatePost'],
                ],
                [
                    'allow' => true,
                    'actions' => ['delete'],
                    'roles' => ['deletePost'],
                ],
            ],
        ],
    ];
```

```
}
```

If all the CRUD operations are managed together then it's a good idea to use a single permission, like managePost, and check it in *yii2\web\Controller::beforeAction()*.

Working with Passwords

Most developers know that passwords cannot be stored in plain text, but many developers believe it's still safe to hash passwords using md5 or sha1. There was a time when using the aforementioned hashing algorithms was sufficient, but modern hardware makes it possible to reverse such hashes and even stronger ones very quickly using brute force attacks. In order to provide increased security for user passwords, even in the worst case scenario (your application is breached), you need to use a hashing algorithm that is resilient against brute force attacks. The best current choice is bcrypt. In PHP, you can create a bcrypt hash using the crypt function. Yii2 provides two helper functions which make using crypt to securely generate and verify hashes easier. When a user provides a password for the first time (e.g., upon registration), the password needs to be hashed:

```
$hash = Yii2::$app->getSecurity()->generatePasswordHash($password);
```

The hash can then be associated with the corresponding model attribute, so it can be stored in the database for later use. When a user attempts to log in, the submitted password must be verified against the previously hashed and stored password:

```
if (Yii2::$app->getSecurity()->validatePassword($password, $hash)) {
    // all good, logging user in
} else {
    // wrong password
}
```

§4. Developing the virtual reception system

1. Installing Yii2 from an archive file

Installing Yii2 from an archive file involves three steps:

- Download the archive file from *yii2framework.com.*
- Unpack the downloaded file to a Web-accessible folder.
- Modify the *config/web.php* file by entering a secret key for the *cookieValidationKey* configuration item (this is done automatically if you are installing Yii2 using Composer):

```
// !!! insert a secret key in the following (if it is empty) - this is required by cookie validation
'cookieValidationKey' => 'enter your secret key here',
```

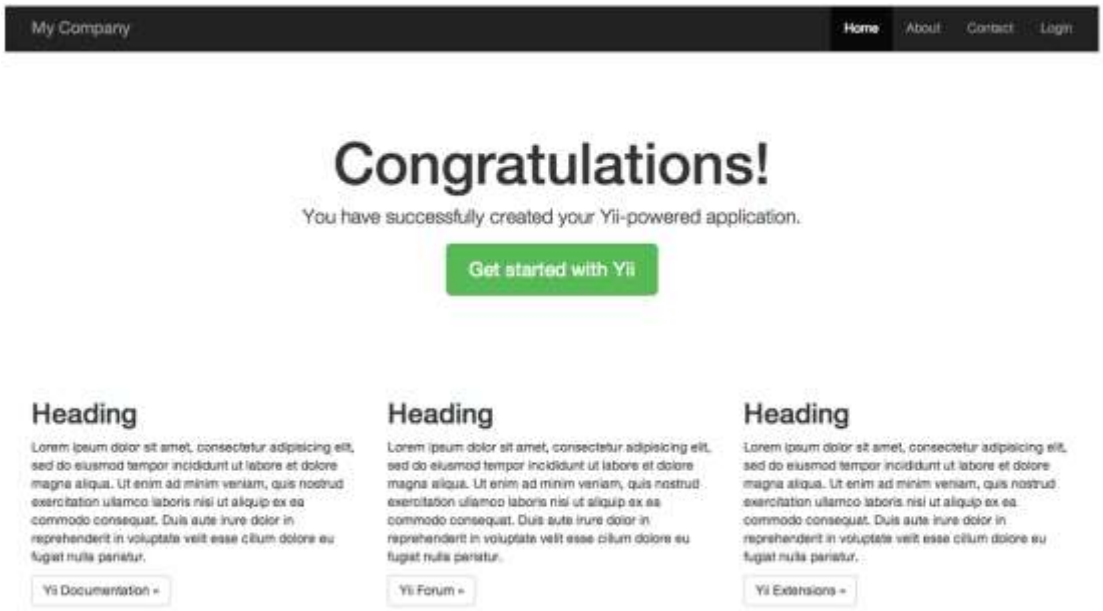
Verifying the Installation

After installation is done, either configure your web server (see next section) or use the built-in PHP web server by running the following console command while in the project web directory:

```
php yii serve --port=8888
```

You can use your browser to access the installed Yii2 application with the following URL:

```
http://localhost:8080/
```



2. Preparing the database

To begin, create a database named **virtualres**, from which you will fetch data in your application. We may create an MySQL database, as Yii2 has built-in support for many database applications.

Next, create a table named ***xatlar*** in the database. We may run the following SQL statements to do so:

```
CREATE TABLE `xatlar` (
  `id` int(11) NOT NULL,
  `fish` varchar(255) CHARACTER SET utf8 NOT NULL,
  `hudud` varchar(255) CHARACTER SET utf8 NOT NULL,
  `tuman` varchar(255) CHARACTER SET utf8 NOT NULL,
  `manzil` varchar(255) CHARACTER SET utf8 NOT NULL,
  `email` varchar(50) CHARACTER SET utf8 NOT NULL,
  `tel` varchar(20) CHARACTER SET utf8 NOT NULL,
  `jins` varchar(255) CHARACTER SET utf8 NOT NULL,
  `tugilgan_yil` varchar(255) CHARACTER SET utf8 NOT NULL,
  `maqom` varchar(255) CHARACTER SET utf8 NOT NULL,
  `mur_qis_mazm` varchar(255) CHARACTER SET utf8 NOT NULL,
  `fayl` varchar(255) CHARACTER SET utf8 NOT NULL,
  `mur_matni` text CHARACTER SET utf8 NOT NULL,
  `mur_raq` int(11) NOT NULL,
  `mur_cod` int(11) NOT NULL,
  `vaqt` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Next, create a table named ***juwapl*** in the database:

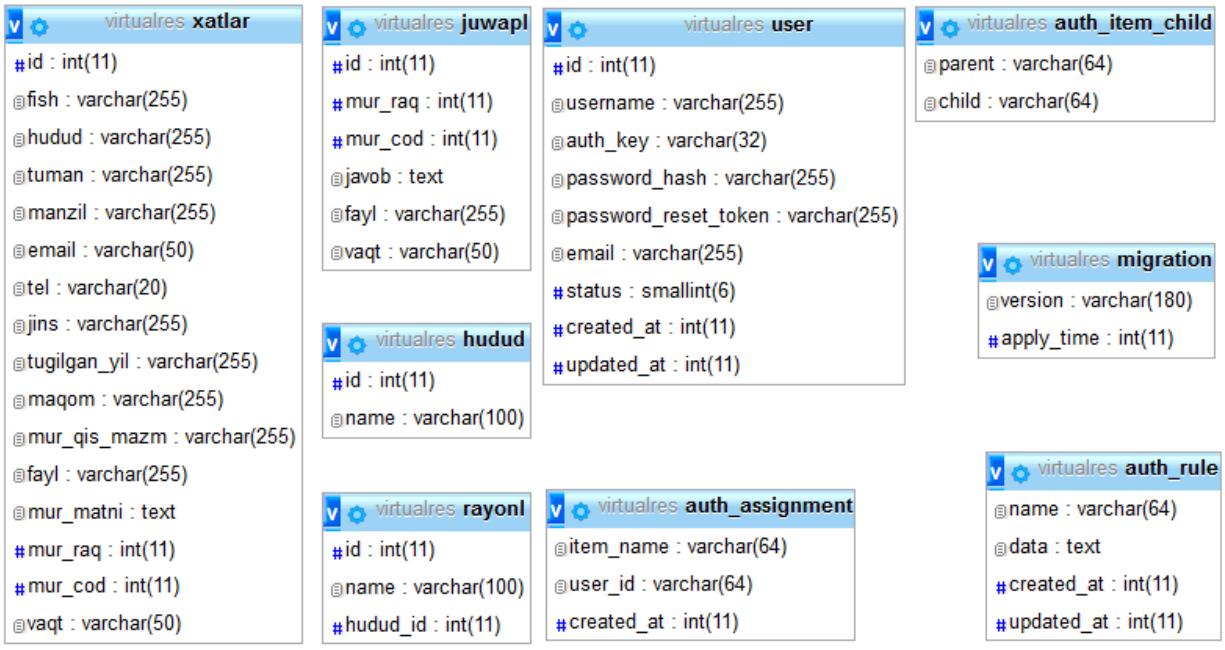
```
CREATE TABLE `juwapl` (
  `id` int(11) NOT NULL,
  `mur_raq` int(11) NOT NULL,
  `mur_cod` int(11) NOT NULL,
  `javob` text CHARACTER SET utf8 NOT NULL,
  `fayl` varchar(255) CHARACTER SET utf8 NOT NULL,
  `vaqt` varchar(50) CHARACTER SET utf8 NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Next, create a table named ***hudud*** in the database:

```
CREATE TABLE `hudud` (
  `id` int(11) NOT NULL,
  `name` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `hudud` (`id`, `name`) VALUES
(1, 'Andijon viloyati'),
(2, 'Buxoro viloyati'),
(3, 'Farg\'ona viloyati'),
(4, 'Jizzax viloyati'),
(5, 'Xorazm viloyati'),
(6, 'Namangan viloyati'),
(7, 'Navoiy viloyati'),
(8, 'Qashqadaryo viloyati'),
(9, 'Qoraqalpog\'iston Respublikasi'),
(10, 'Samarqand viloyati'),
(11, 'Sirdaryo viloyati'),
(12, 'Surxondaryo viloyati'),
(13, 'Toshkent viloyati'),
(14, 'Toshkent shahri');
```

Next, create a table named ***rayonl*** in the database:

```
CREATE TABLE `rayonl` (
  `id` int(11) NOT NULL,
  `name` varchar(100) NOT NULL,
  `hudud_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```



Configuring a data base connection.

Before proceeding, make sure you have installed both the PDO PHP extension and the PDO driver for the database you are using. This is a basic requirement if your application uses a relational database. With those installed,

open the file *../config/db.php* and change the parameters to be correct for your database. By default, the file contains the following:

```
<?php

return [
    'class' => 'yii\db\Connection',
    'dsn' => 'mysql:host=localhost;dbname=virtualres',
    'username' => 'root',
    'password' => '',
    'charset' => 'utf8',
];
```

Generating code with Gíi.

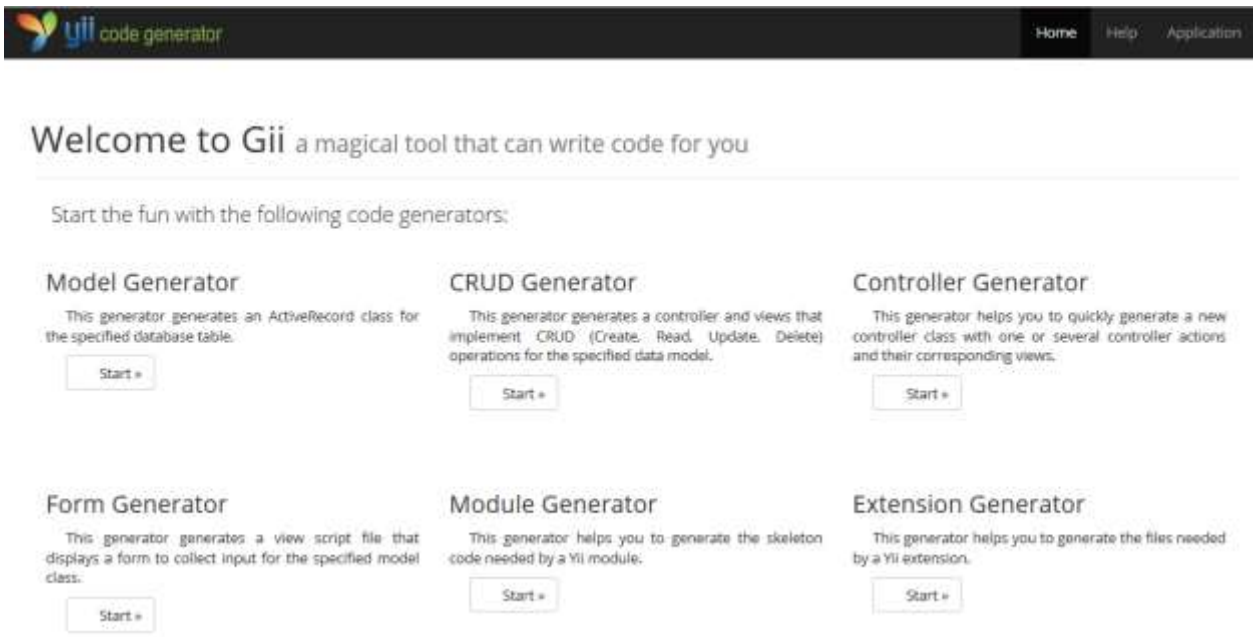
This section will describe how to use Gíi to automatically generate code that implements some common Web site features. Using Gíi to auto-generate code is simply a matter of entering the right information per the instructions shown on the Gíi Web pages. Through this tutorial, you will learn how to:

- enable Gíi in your application,
- use Gíi to generate an Active Record class,
- use Gíi to generate the code implementing the CRUD operations for a DB table,
- customize the code generated by Gíi.

Starting Gíi**.**

We can now access Gíi via the following URL:

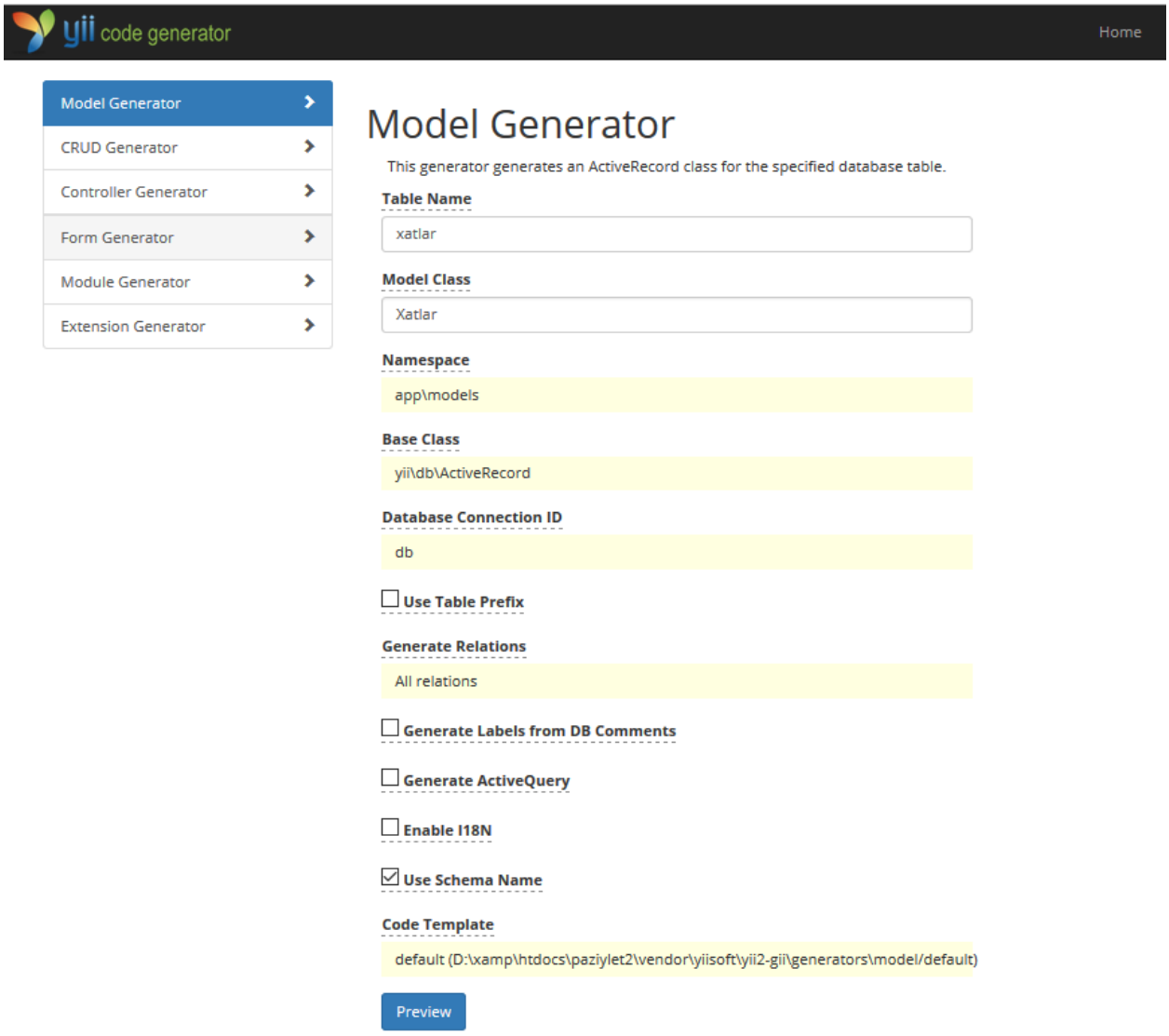
```
http://hostname/index.php?r=gii
```



Generating an Active Record Class.

To use Gíi to generate an Active Record class, select the "Model Generator" (by clicking the link on the Gíi index page). Then fill out the form as follows:

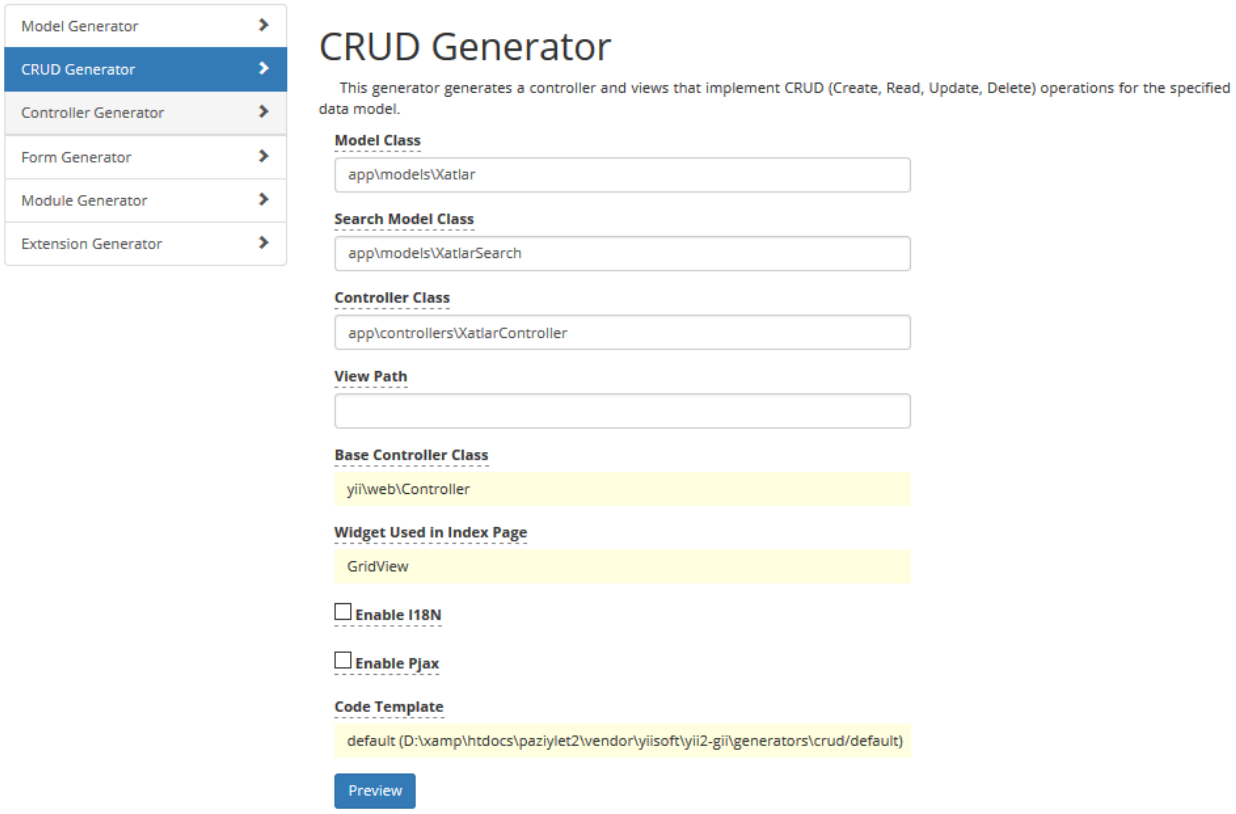
- Table Name: xatlar
- Model Class: Xatlar



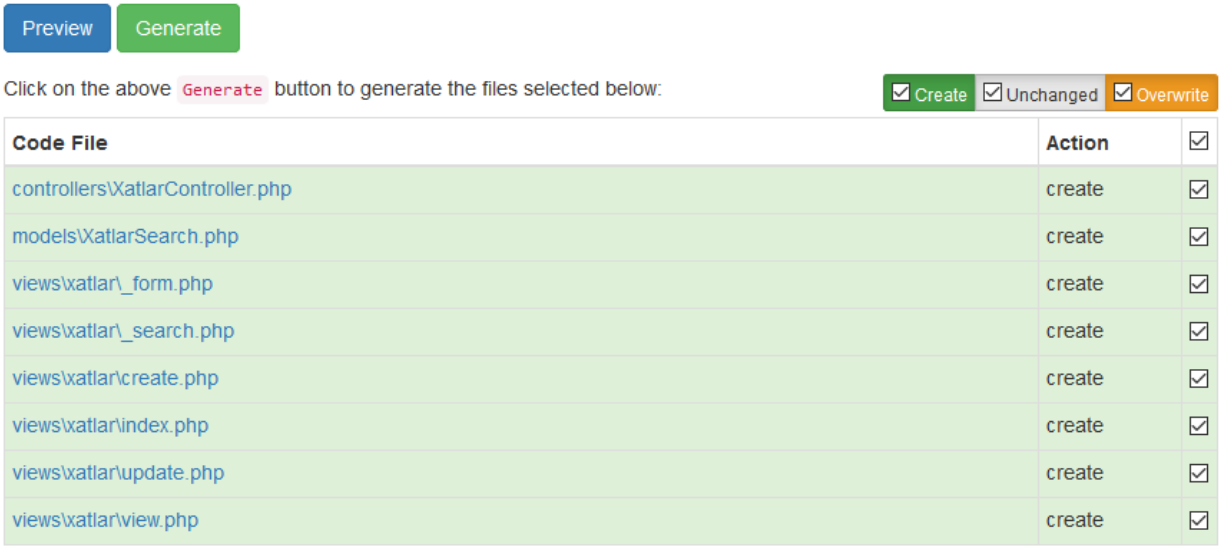
Generating CRUD code.

CRUD stands for Create, Read, Update, and Delete, representing the four common tasks taken with data on most Web sites. To create CRUD functionality using Gíi, select the "CRUD Generator" (by clicking the link on the Gíi index page). For the *"xatlar"* example, fill out the resulting form as follows:

- Model Class: app\models\Xatlar
- Search Model Class: app\models\XatlarSearch
- Controller Class: app\controllers\XatlarController



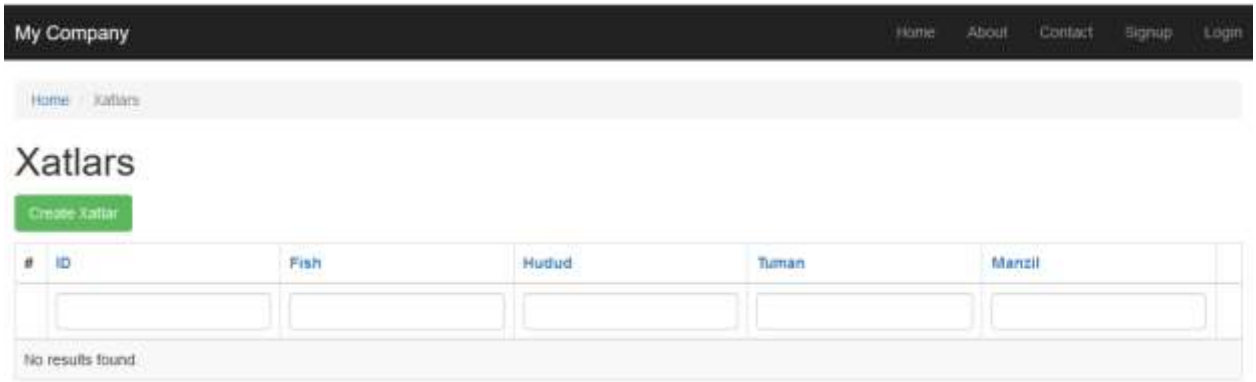
Next, click on the "*Preview*" button. You will see a list of files to be generated, as shown below.



Trying it out.

To see how it works, use your browser to access the following URL:

```
http:http://localhost/virtualres/frontend/web/index.php?r=xatlar
```



In this section, you have saw how to use Gíi to generate the code that implements complete CRUD functionality for content stored in a database table.

Installing the new theme AdminLTE.

AdminLTE - is a fully responsive admin template. Based on Bootstrap 3 framework. Highly customizable and easy to use. Fits many screen resolutions from small mobile devices to large desktops. Check out the live preview now and see for yourself.

This package contains an Asset Bundle for Yii22 Framework which registers the CSS files for the AdminLTE user-interface. The CSS files are installed via Yii2's recommended usage of the fxp/composer-asset-plugin v1.1.1 or later.

Installation.

The preferred way to install this extension is through composer.

To install AdminLTE v2 run:

```
php composer.phar require dmstr/yii22-adminlte-asset "2.*"
```

To install AdminLTE v1 run:

```
php composer.phar require dmstr/yii22-adminlte-asset "1.*"
```

Quick Start.

Once the extension is installed, you can have a preview by reconfiguring the path mappings of the view component: For Yii2 2 Advanced Application Template or Basic Application Template

```
'components' => [
    'view' => [
         'theme' => [
             'pathMap' => [
                '@app/views' => '@vendor/dmstr/yii2-adminlte-asset/example-views/yiisoft/yii2-app'
             ],
         ],
    ],
],
```

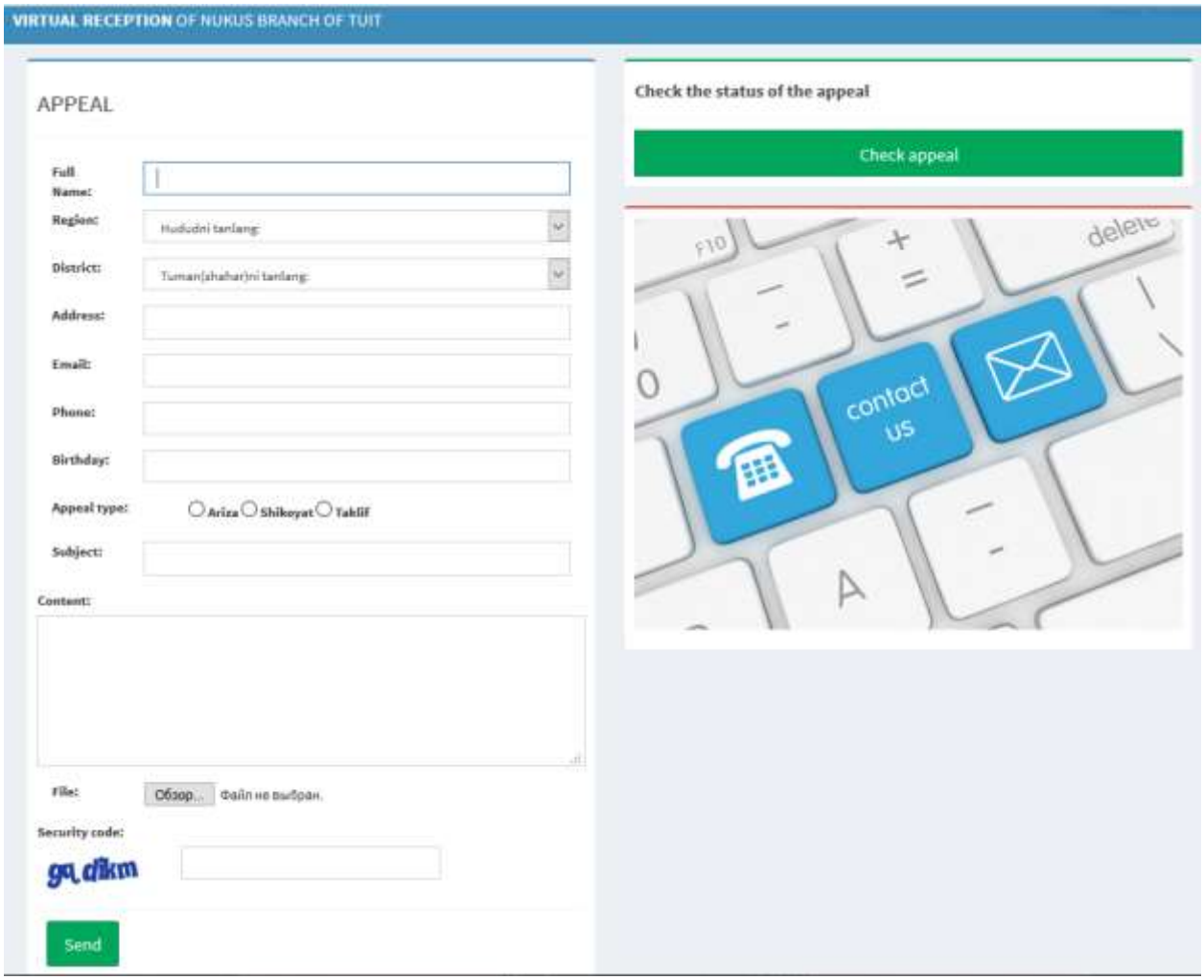
This asset bundle provides sample files for layout and view (see folder examples/), they are not meant to be customized directly in the vendor/ folder. Therefore it is recommended to copy the views into your application and adjust them to your needs.

§5. About virtual reception system of the Nukus branch of TUIT

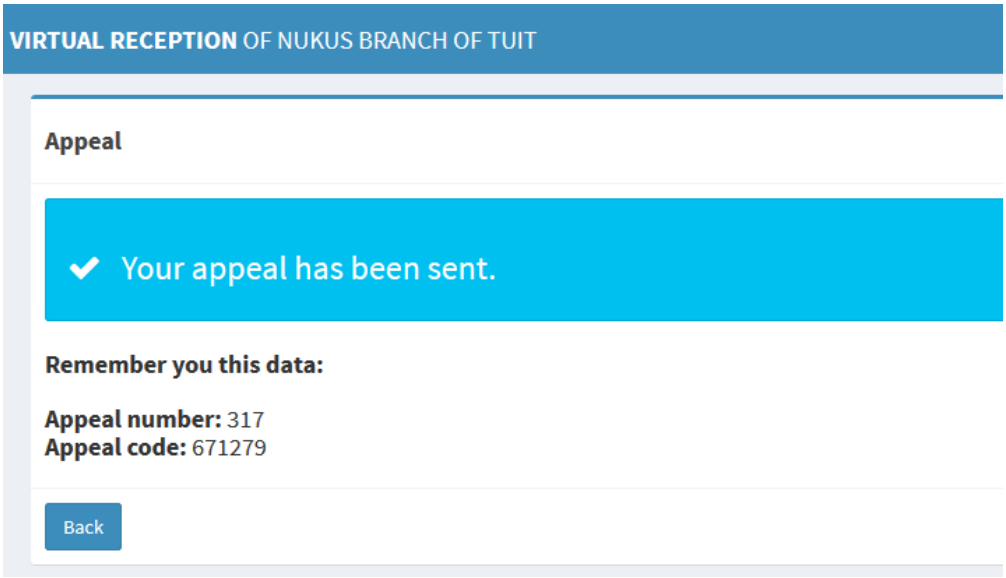
Virtual reception of branch is consist of this parts:

- Part of sending appeal (frontend),
- Part of checking the appeals (backend).

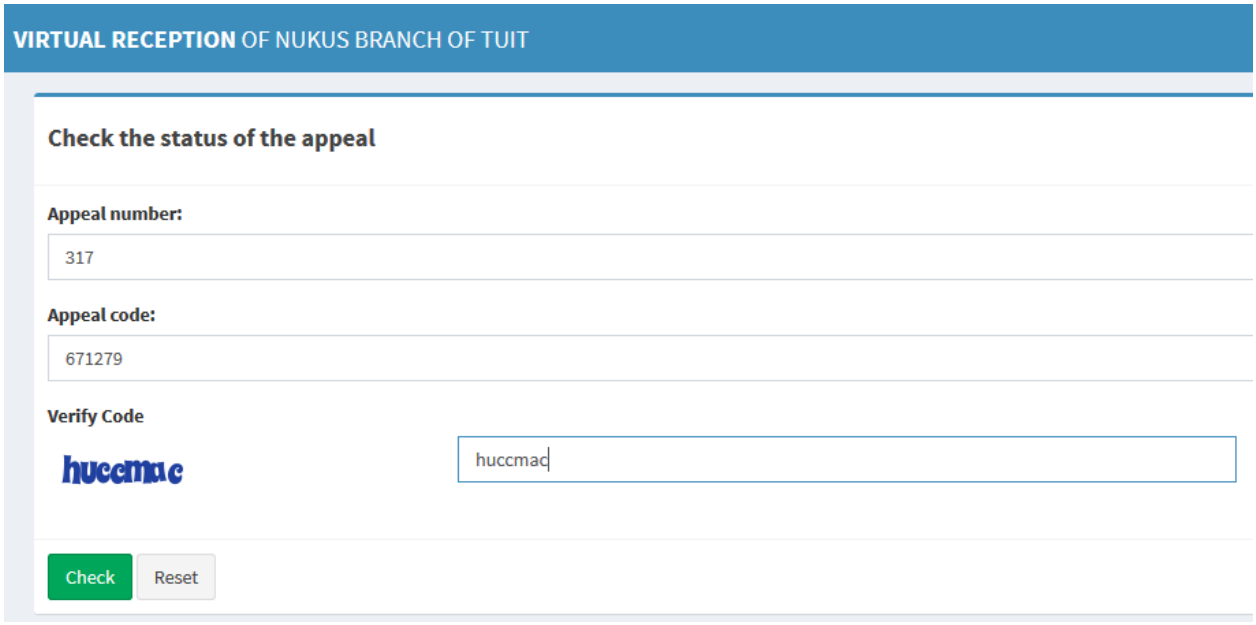
Part of sending appeal:



In this part user have filled this application and will send. After sent this appeal I can see this dialog window:



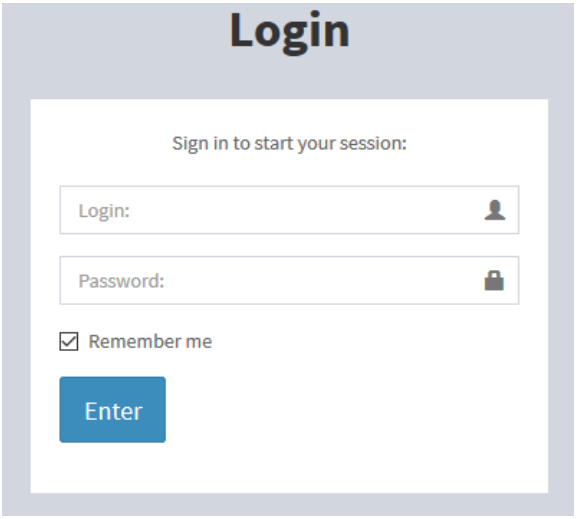
User can know own appeals result through pass the button “Check the status of the appeal”:



After insert the “Appeal number”, “Appeal code” and “Security code” pass the button “Check”.

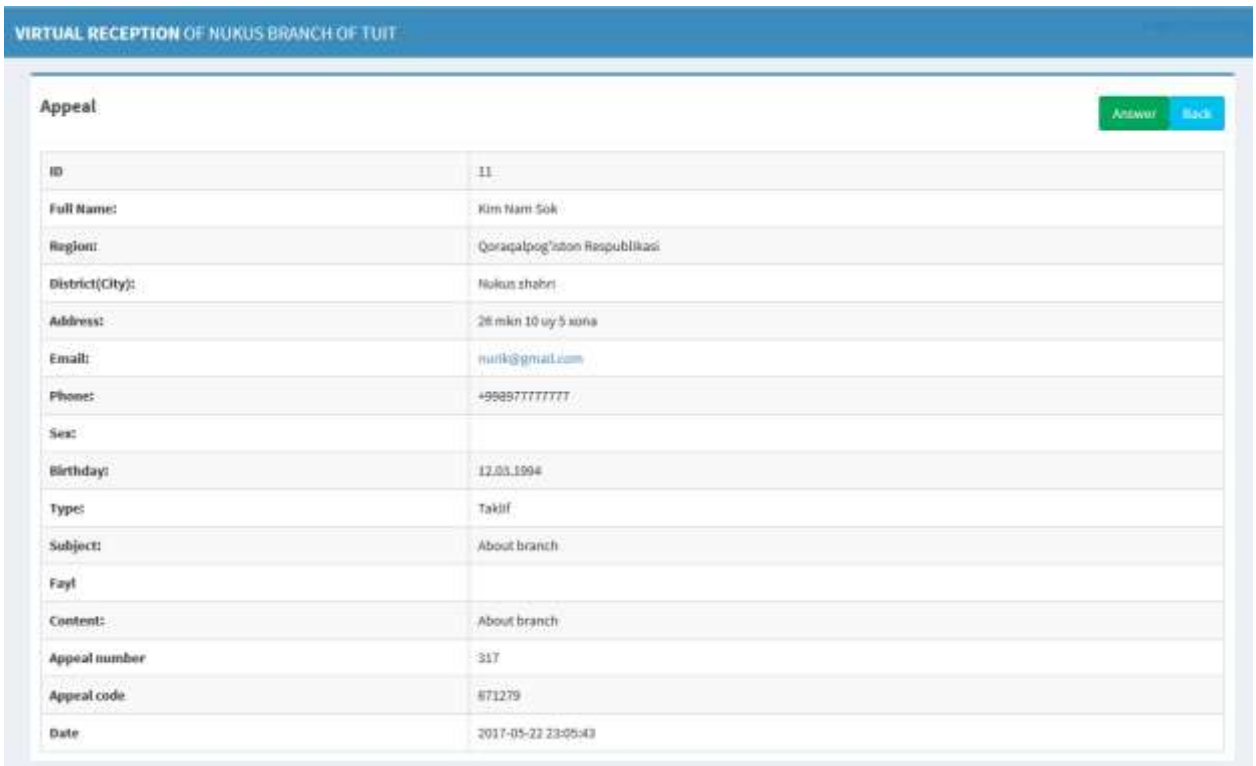
Part of checking the appeals.

To implement entry the this part just through system administrator, so give the special login and password to the system administrator and entered through this login and password.

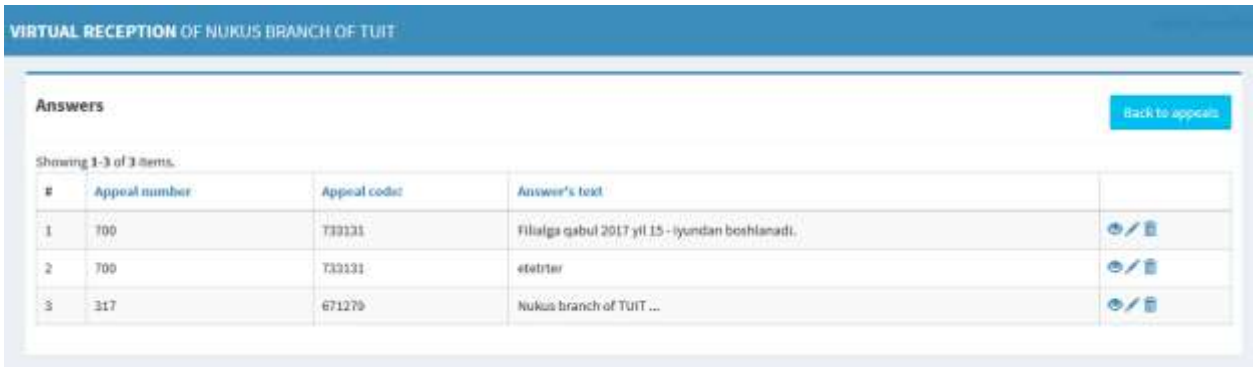


After entered the part admin system we can see list of appeals arrive the "Virtual reception" system.

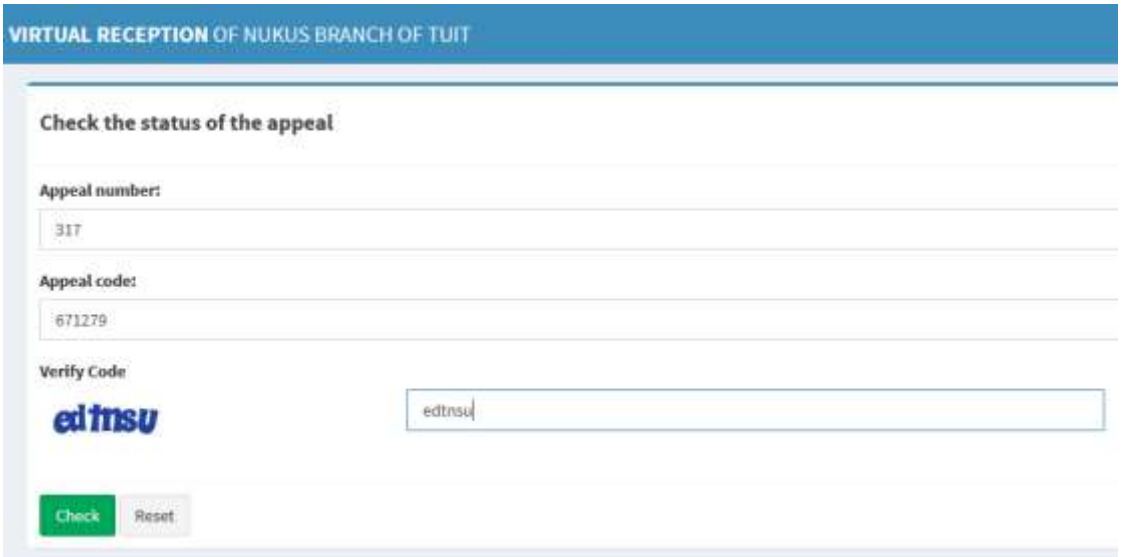
After pass button "View appeal" we can see full content about this appeal.



To write to the arrived appeal through to pass the button “Answer” and we will entered it to the database through pass the button “Send”.



The owner this appeal give the branch directorate’s answer through pass the button “Check appeal”.



Content of answer:

Conclusion

Dedicated to Developing the virtual reception system of the Nukus branch of TUIT using Yii framework, this qualification paper aims to automatize organizing direct communication with people, receiving electron version of the appeals of physical and juridical people and answering them on time through this system in the example of the Nukus branch of TUIT.

Yii is the most powerful PHP framework which was the sole reason for the attention it caught as soon as it got a stable release. This open source web application development framework has impressive performance. The popularity of Yii framework is increasing at a constant pace and once used, Yii is always preferred again for “Yes It Is” a power packed Framework.

Like any good framework, Yii helps you create modern web applications quickly, and make sure they perform well. It pushes you to create secure and testable sites by doing a lot of the heavy lifting for you. You can easily use most of its features exactly as they are provided, or you can modify each one to suit your needs. I really encourage you to check it out for your next web project.

In this qualification paper controlling in the system of MySQL information base virtual reception system is created and virtual reception system is created by using Yii 2 PHP framework.

References