



### Core Concepts & Architecture

#### MVC Structure

<b>Model</b>	Represents data and business logic. Interacts with the database.
<b>View</b>	Presents the data to the user. Consists of HTML, CSS, and PHP code for display.
<b>Controller</b>	Handles user requests, interacts with models, and selects views to render.
<b>Entry Script (index.php)</b>	The single entry point for all web requests. Initializes the application.
<b>Application</b>	The central object that manages the overall execution flow.
<b>Components</b>	Reusable modules providing specific functionalities (e.g., database, session, user).

#### Application Lifecycle

1. User makes a request (e.g., `index.php?r=post/view&id=123`).
2. Entry script (`index.php`) creates and initializes the application.
3. Application retrieves request information from request component.
4. Application creates a controller instance to handle the request.
5. Controller creates action instance and performs the action.
6. Action loads relevant data models, possibly with database interaction.
7. Action renders a view, passing the models as parameters.
8. View renders the data into HTML.
9. The rendered result is returned to the user.

#### Configuration

<b>Configuration Array</b>	Yii applications are configured using a PHP array, typically located in <code>config/web.php</code> or <code>config/console.php</code> .
<b>Components Configuration</b>	Configures core application components such as <code>db</code> , <code>cache</code> , <code>user</code> , <code>session</code> , etc.
<b>Modules Configuration</b>	Defines modules and their specific configurations.
<b>Parameters Configuration</b>	Defines global application parameters accessible throughout the application.
<b>Example</b>	<pre>'components' =&gt; [     'db' =&gt; [         'class' =&gt;             'yii\db\Connection',         'dsn' =&gt;             'mysql:host=localhost;dbname=             mydatabase',         'username' =&gt; 'root',         'password' =&gt; '',         'charset' =&gt; 'utf8',     ], ],</pre>

### Database Interaction

#### Active Record

Active Record (AR) provides an object-oriented interface for accessing and manipulating data stored in databases. Each AR class represents a database table, and an AR instance represents a row in that table.
<b>Defining an AR Class</b> <pre>class Customer extends \yii\db\ActiveRecord {     public static function tableName()     {         return 'customers';     } }</pre>
<b>Basic CRUD Operations</b> <ul style="list-style-type: none"> <li>• <b>Create:</b> <code>\$customer = new Customer(); \$customer-&gt;name = 'John Doe'; \$customer-&gt;email = 'john.doe@example.com'; \$customer-&gt;save();</code></li> <li>• <b>Read:</b> <code>\$customer = Customer::findOne(123);</code> or <code>\$customers = Customer::findAll(['status' =&gt; 1]);</code></li> <li>• <b>Update:</b> <code>\$customer = Customer::findOne(123); \$customer-&gt;email = 'new.email@example.com'; \$customer-&gt;save();</code></li> <li>• <b>Delete:</b> <code>\$customer = Customer::findOne(123); \$customer-&gt;delete();</code></li> </ul>

#### Query Builder

The Query Builder provides a programmatic and database-agnostic way to construct SQL queries.
<b>Example:</b> <pre>\$customers = (new \yii\db\Query())     -&gt;select(['id', 'name', 'email'])     -&gt;from('customers')     -&gt;where(['status' =&gt; 1])     -&gt;orderBy('name')     -&gt;limit(10)     -&gt;all();</pre>
<b>Chaining Methods:</b> The Query Builder allows you to chain methods to build complex queries easily.

## Migrations

<b>Creating a Migration</b>	<code>./yii migrate/create create_users_table</code>
<b>Applying Migrations</b>	<code>./yii migrate</code>
<b>Reverting Migrations</b>	<code>./yii migrate/down</code>
<b>Migration Class Structure</b>	<pre>class m150101_185401_create_users_table extends \yii\db\Migration { public function up() { \$this-&gt;createTable('users', [ 'id' =&gt; \$this-&gt;primaryKey(), 'username' =&gt; \$this-&gt;string()-&gt;notNull()-&gt;unique(), 'email' =&gt; \$this-&gt;string()-&gt;notNull()-&gt;unique(), ]); } public function down() { \$this-&gt;dropTable('users'); } }</pre>

## Working with Views & Controllers

### Rendering Views

<b>Rendering a Simple View</b>	<code>\$this-&gt;render('view', ['model' =&gt; \$model]);</code>
<b>Rendering a View with Layout</b>	<code>\$this-&gt;render('view', ['model' =&gt; \$model, 'main'];</code>
<b>Rendering a Partial View</b>	<code>\$this-&gt;renderPartial('_form', ['model' =&gt; \$model]);</code>
<b>Accessing Variables in Views</b>	Variables passed to the <code>render()</code> method are available in the view as local variables (e.g., <code>\$model</code> ).

### Controller Actions

Controller actions are methods within a controller class that handle specific user requests. They typically perform tasks such as loading data, processing user input, and rendering views.
<b>Action Naming Convention:</b> Action names should start with the word <code>action</code> (e.g., <code>actionCreate</code> , <code>actionView</code> ).
<b>Example:</b> <pre>public function actionView(\$id) { \$model = \$this-&gt;findModel(\$id); return \$this-&gt;render('view', ['model' =&gt; \$model]); }</pre>

### Layouts

<b>Main Layout</b>	The default layout file, typically located in <code>views/layouts/main.php</code> , defines the overall structure of the web page.
<b>Layout Structure</b>	Layout files typically contain HTML <code>&lt;html&gt;</code> , <code>&lt;head&gt;</code> , and <code>&lt;body&gt;</code> tags, as well as placeholders for content and other dynamic elements.
<b>Rendering Content in Layout</b>	The <code>\$content</code> variable within the layout file holds the rendered output of the view.

## Forms and Input Validation

## Creating Forms

Forms in Yii are typically created using the `yii\widgets\ActiveForm` widget, which simplifies the process of generating HTML form elements and handling user input.

### Example:

```
<?php $form = ActiveForm::begin(['id' =>
'login-form']); ?>

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

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

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

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

## Input Validation

**Validation Rules** Define validation rules in the model's `rules()` method. Rules specify which attributes should be validated and how.

**Common Validators** `required`, `email`, `string`, `integer`, `number`, `boolean`, `date`, `unique`, `exist`, `captcha`.

**Example:**

```
public function rules()
{
    return [
        [['username',
'password'], 'required'],
        ['email', 'email'],
        ['username', 'string',
'min' => 3, 'max' => 255],
    ];
}
```

## Handling Form Submission

In the controller action, check if the form has been submitted and if the model is valid. If so, process the data and redirect the user.

### Example:

```
public function actionLogin()
{
    $model = new LoginForm();
    if ($model->load(Yii::$app->request-
>post()) && $model->login() {
        return $this->goHome();
    }

    return $this->render('login', ['model' =>
$model]);
}
```