Yii Framework Cheatsheet

A concise reference guide to the Yii PHP framework, covering core components, commonly used features, and best practices for efficient web application development.

Core Concepts & Architecture

MVC Structure

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

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

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.

Defining an AR Class

```
class Customer extends
\yii\db\ActiveRecord
{
    public static function tableName()
    {
        return 'customers';
    }
}
```

Basic CRUD Operations

- **Create:** `$customer = new Customer(); $customer->name = 'John Doe'; $customer->email = 'john.doe@example.com'; $customer->save();`
- **Read:** `$customer = Customer::findOne(123);` or `$customers = Customer::findAll(['status' => 1]);`
- **Update:** `$customer = Customer::findOne(123); $customer->email = 'new.email@example.com'; $customer->save();`
- **Delete:** `$customer = Customer::findOne(123); $customer->delete();`

Query Builder

The Query Builder provides a programmatic and database-agnostic way to construct SQL queries.

Example:

```
$customers = (new \yii\db\Query())
    ->select(['id', 'name', 'email'])
    ->from('customers')
    ->where(['status' => 1])
    ->orderBy('name')
    ->limit(10)
    ->all();
```

Chaining Methods: The Query Builder allows you to chain methods to build complex queries easily.

Migrations

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

Working with Views & Controllers

Rendering Views

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

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.

Action Naming Convention: Action names should start with the word `action` (e.g., `actionCreate`, `actionView`).

Example:

```
public function actionView($id)
{
    $model = $this->findModel($id);

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

Layouts

Main Layout	The default layout file, typically located in `views/layouts/main.php`, defines the overall structure of the web page.
Layout Structure	Layout files typically contain HTML `<html>`, `<head>`, and `<body>` tags, as well as placeholders for content and other dynamic elements.
Rendering Content in Layout	The `$content` 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]);
}
```