CodeIgniter 4

CodeIgniter 4 From Scratch

#1 - Introduction

#2 - Local Environment Setup | XAMPP Installation | Composer

https://codeigniter.com/user_guide/installation/index.html

● Yeni bir CodeIgniter projesi oluşturmak için komut satırına;

```
composer create-project codeigniter4/appstarter projeadi
```

● Uygulama /public klasöründe içinden başlıyor.

● localhost'u projenin ana sayfası yapmak için;
XAMMP > Manage Servers > Apache Web Server > Configure > Open Conf File
Açılan httpd.conf dosyasında DocumentRoot kısmındaki adresleri projenin adresine ayarlamamız yeterli;
DocumentRoot "/opt/lampp/htdocs"
<Directory "/opt/lampp/htdocs">

● /public klasörü localhost üzerinden açılmıyorsa .htaccess dosyasını iptal et. Hataları gider...
* apache hata log'larına bak! moderewrite ile ilgili bir sıkıntı var (Invalid command '<IfModule')
veya
● CodeIgniter'ın kendi php sunucusunu çalıştırmak için;

```
php spark serve
```

#3 - Folder Structure Overview

https://codeigniter.com/user_guide/concepts/structure.html

● /public klasöründe hiçbir değişiklik veya ekleme yapmayacağız. Yapacağımız tek ekleme asset dosyaları olacak.
Resimler için /public/assets/images klasörünü oluşturacağız.

● Projeye başlamadan önce **env** dosyasını .env olarak kopyalamalıyız. Bu bizim proje ayarları dosyamın olacak.
● Güvenlik nedeniyle Codegniter'da hata mesajları gösterilmiyor. Geliştirme yaparken hataları görebilmek için .env dosyasındaki

```
# CI_ENVIRONMENT = production
```

satırını

```
CI_ENVIRONMENT = development
```

olarak değiştiriyoruz.
Ayrıca base URL'imizi ayarlıyoruz (Spark server için 'http://localhost:8080/');

```
app.baseURL = 'http://localhost/'
```

● Veritabanı ayarları için DATABASE bölümüne bakacağız. Default; kullanılan veritabanı. Tests; unit test veritabanı.

● /writable klasöründe CodeIgniter'a ait depolanan cache, cookie gibi veriler yer alıyor. Bu klasörle çok işimiz olmayacak.

● /tests ile de çok işimiz olmayacak.

● /vendor klasöründe composer ile yüklenen bilşenler yer alıyor. Buradaki dosya ve kalsörleri değiştirmeyeceğiz.

● **/app** klasörü içinde çalışacağız.

● /app**/Config** içindeki dosyalarla, kullanılan kütüphanlerin ayarlarını yapabiliriz.
Migrations, Validation, Routes gibi.

● /app/Config**/Routes.php** dosyasından default controller ve controller için default methodu ayarlayabiliriz.

● Yeni bir validation kuralı oluşturduğumzda /app/Config**/Validation.php** dosyasında bunu tanımlamalıyız.

#4 - Controllers

https://codeigniter.com/user_guide/incoming/controllers.html

● /app**/Controllers** klasörü içinde controller dosyalarımız var.

● Ne zaman bir controller çağırılsa **index()** metodu default olarak çalışır.

● View'a parametre aktarmak için (URL'den) ilgili controllerin metoduna parametre olarak vermeliyiz.
```
public function metod($param1, $param2, ...) { ... }
```

● http://localhost:8080/controllerAdı/methodAdı/parametre1/parametre2 mantığıyla çalışıyor.

● Eğer metoda url'den doğrudan erişilmesin istiyorsak metodu protected olarak tanımlamalıyız. Örenğin bir login veya validation metodu gibi.

● Eğer /Controllers klasörü altında özel bir klasör yapısı/alt kategoriler oluşturmak istiyorsak kalsörlerin adı büyük harfle başlamalı, alt klasörde yer alan dosyaların namespace'ini o klasöre uygun olarak değiştirmeli ve BaseController'ı kullanabilmek için use ile dahil etmeliyiz.
```
namespace App\Controllers\AltKlasorAdı;
use App\Controllers\BaseController;
```

http://localhost:8080/klasorAdı/controllerAdı/methodAdı/parametre1/parametre2

● Bir controller'dan başka bir controller'a ait bir metodu çağırmak için o controller'ın nesnesini oluşturmalıyız.
Başka bir klasördeki controller'ın metodunu kullanmak için önce o controller'ın namespace'ini use ile tanımlamalıyız. İsim çakışması olmasın diye as ile takma isim atayabiliriz.
```
use App\Controllers\KlasorAdı\ControllerAdı as TakmaAd;
// ... Metod içine ...
$takmaAd = new TakmaAd();
$takmaAd->metod("param1", "param2", ...);
```

#5 - Routes

https://codeigniter.com/user_guide/incoming/routing.html

● Default controller ve metod ataması için Routes.php dosyasında;
```
$routes->setDefaultController('ControllerDosyası');
$routes->setDefaultMethod('metod');
```

● Özel route ve url'ler yaratmak için Routes.php dosyasında;
```
$routes->add('urlAdı', 'ControllerAdı::metod');
```
Artık metoda http://localhost:8080/urlAdı ile erişebiliriz.

● Eğer parametre kabul etsin istiyorsak/kabul ediyorsa her parametre için;
```
$routes->add('urlAdı/(:any)/(:any)', 'ControllerAdı::metod/$1/$2');
```
http://localhost:8080/urlAdı/param1/param2

● Anonymous fonksiyonlar ile de url tanımlayabiliriz;
```
$routes->add('urlAdı', function() {
        // ...
});
```

● Url grupları oluşturabiliriz. İlk bölümüne göre tanımlı olurlar. group() metodunda tanımlanan isimle başlayan url'ler bu kuralla işlenir.
Klasör\Dosya yolu setDefaultNamespace('App\Controllers') 'a göre belirlenir.
```
$routes->group('1.url', function($routes) {
        $routes->add('2.url', 'Klasör\Dosya::metod');
        $routes->add('2.url', 'Klasör\Dosya::metod');
        $routes->add('2.url/(:any)/(:any)', 'Klasör\Dosya::metod/$1/$2');
});
```
http://localhost:8080/1.url/2.url/varsaParam1/varsaParam2

● Route oluşturabilmenin başka bir önemli özelliği de yapılan HTTP request'ine (POST, GET veya PUT, PATCH vs) göre özel route tanımlayabilmektir.

● get() will work only for GET requests. But if you will submit a form with POST method on that same url, get() route will not be triggered.

Örnek:
blog_from.php formumuz;
```
<form action="/admin/blog/new" method="post"> ... </form>
// action="/varsaAltKlasör/dosya/metod"
```
Admin altklasöründeki Blog.php controller'ımız;
```
...
public function index() { // List of blog posts... }
public function createNew() { return view('blog_form'); }
public function saveBlog() { // formun işlenmesi, validate'i }
```

Routes.php'de tanımladığımız özel route'lar;
```
...
$routes->group('admin', function($routes) {
        $routes->add('blog', 'Admin\Blog::index'); // http:///localhost:8080/admin/blog
        $routes->get('blog/new', 'Admin\Blog::createNew');
        $routes->post('blog/new', 'Admin\Blog::saveBlog');
});
```

● Aynı zamanada get ve post metodlarını aynı route içinde kullanabiliriz;
```
$routes->match(['get', 'post'], 'url', 'Controller::method');
```

#6 - Views | Basic Views | Part 1/3
https://codeigniter.com/user_guide/outgoing/views.html

● View dosyaları app/**Views** içinde bulunur.

● **view('varsaAltKlasör/viewDosyası', $varsaParam)** metoduyla view render'larız.

● View'a veri geçirmek için view() metoduna parametre olarak vermeliyiz. İlişkisel dizi olarak verlien parametre, ilgili view dosyasından **<?= ?>** etiketi (echo) içinde indeks adıyla çağırılır.

#7 - Views | View Cells | Part 2/3

https://codeigniter.com/user_guide/outgoing/view_cells.html

● Daha temiz bir görünüm için view'ları component'ler şeklinde oluşturabiliriz ve view_cell() metodunu kullanarak görüntüleyebiliriz. Böylece oluşturduğumuz bir component'i başka bir yerde de view_cell() matodunu çağırarak kullanabiliriz.

● Öncelikle View/components klasörünü oluştururuz ve components altında component view dosyamızı oluşturuz. Varsa yardımcı sınıftaki metoddan gelen verileri kullanabiliriz.
● Daha sonra App**/Libraries** klasörü altında yardımcı sınıfımızı oluştururuz;

```
namespace App\Libraries;
class YardımcıSınıf {
        public function metodAdı($params) { // view'dan gelen param
                return view('components/view_dosyası', $params);
        }
}
```

● Asıl view dosyamızda (controller'da render'lanan) **view_cell()** metoduyla component'ımızı çağırır ve kullanırız;

```
view_cell('App\Libraries\YardımcıSınıf::metodAdı', ['params' => $controllerdanGelenParam]);
```

#8 - Views | View Layouts | Part 3/3

https://codeigniter.com/user_guide/outgoing/view_layouts.html

● View dosyalarımızda kolayca layout kullanabilmek için; View/layouts klasörünü oluşturduk.
● Layouts klasörü altında header, footer ve gerekli etiketlerden oluşan dosyamızı oluşturduk.
Header ve footer arasındaki içerik kısmına;

```
$this->renderSection('isim')
// Daha sonra view dosyalarımızda aynı ismi kullanmamız koşuluyla verdiğimiz 'isim'  farketmez.
```

● Daha sonra bu oluşturduğumuz layout'u kullanmak istediğimiz view'a;

```
$this->extend('layouts/layoutAdı'); // başa
$this->section('isim'); // başa
// ... içerik ...
$this->endSection(); // sona
```

● View'lara ayrı bir view include etmek için;

```
$this->include('varsaAltKlasör/viewAdı'); // View klaösrüne göre
```

#9 - Models

https://codeigniter.com/user_guide/incoming/controllers.html

● CodeIgniter\Database\Exceptions\DatabaseException #8 - Unable to connect to the database.
Spark server'da veritabanı işlemlerini yapılabilmesi için writable klasörünün modunu 777 yap
ve
.env'deki database.default.hostname'i 127.0.0.1 yap.
writable folder and child folders must be owned by web server process, in my Linux Centos is apache (production)

```
chown -R apache /var/www/myproject/writable
```

Or allowing total control to everyone (development)

```
chmod 777 -R /var/www/myproject/writable
```

● CodeIginter bize 2 tip model sunuyor.
1.'si varsolan model sınıfını extend eden model. Bize pek çok yardımcı metod sunar.
2.'si bağımsız modeller. Bunları ve içindeki metodları kendimiz yaratırız.

● Veritabanına veri eklemek için;

Veritabanımızı oluşturduk.

.env dosyasında DATABASE bölümünü ayarladık;

```
database.default.hostname = 127.0.0.1 // localhost için
database.default.database = ci4            // db adı
database.default.username = root
database.default.password =
database.default.DBDriver = MySQLi
```

● https://codeigniter4.github.io/userguide/models/model.html adresinden örnek Model'imizi kopyaladık.
Bu model'in oluşturulmasıyla birlikte tüm CRUD metodlarına erişebiliriz.
Extend edilen Model'in doğru çalışması için $table, $primaryKey ve $allowedFields protected alanları gereklidir.
$allowedFields; sınıf dışından değiştirilebilecek alanlar. Örneğin bir kullanıcı formundan.
$useTimestamp, $createdField, $updatedField alanları veritabanındaki ilgili alanları /kayıtları timestamplayabiliriz.

İlgili controller'a oluşturulan modeli dahil ettikten sonra;

```
use App\Models\ModelDosyası;
...
public function metod() {
        if($this->request->getMethod() == 'post') {
                $model = new ModelAdı();
                $model->save($_POST);
        }
//...
}
```

Forma;

```
<form method="post" action="/dosya/metod"> ... </form>
```

● Veritabandan veri çekmek için iligli controller'ın metoduna;

```
public function metod($id) { // $id url'den geliyorsa
        $model = new ModelAdı();
        $obj = $model->find($id);
        // ...
}
```

● Veri silmek için;

```
// ...
$model = new BlogModel();
$obj = $model->find($id);
if($obj) {
        $model->delete($id);
}
return redirect()->to('/viewDosyası');
```

● Güncellemek için;

```
// ...
$model = new BlogModel();
if($this->request->getMethod() == 'post') {
        // save metoduyla update etmek için primary key de verilmelidir
```

```
        $_POST['id'] = $id;
        $model = new ModelAdı();
        $model->save($_POST);
}
```

● Model eventler ile, veritabında işlem yapmdana önce veya yaptıktan sonra olaylar tetikleyebiliriz. Model event'ler; $beforeInsert, $afterInsert, $beforeUpdate, $afterUpdate, $afterFind, $afterDelete

Örnek: Veritabanına kullanıcı şifresi eklemeden önce hashlemek için;
İlgili model dosyasına;

```
protected $beforeInsert = ['hashPassword'];

public function hashPassword(array $data) {
        $data['data']['password_alanı'] = password_hash(
                $data['data']['password_alanı'], PASSWORD_DEFAULT);
        return $data;
}
```

● Özel modeller yaratmak için /Models kalsöründe model dosyamızı oluşturuyoruz ve namespace'ini App\Models yapıyoruz.
Bu sayede özel fonksiyonlar tanımlayabiliriz.

Örnek:

```
namespace App\Models;
use CodeIgniter\Database\ConnectionInterface; // db kullanacaksak

class CustomModel {
        protected $db;

        public function __construct(ConnectionInterface &$db) {
                $this->db = &$db;
        }

        // İki taboyu birleştirdik
        function getPosts() {
                $builder = $this->db->table('posts');
                $builder->join('users', 'posts.post_author = users.user_id');
                $posts = $builder->get()->getResult();
                return $posts;
        }
}
```

● Daha sonra controller'larımızda modeli use ile çağırarak kullanabiliriz;

```
$db = db_connect();
$model = new CustomModel($db);
```

#10 - Query Builder

https://codeigniter.com/user_guide/database/query_builder.html

---

● Otomatik veritabanı doldurmak için; http://filldb.info/

● Sorgular get() metodundan önce yazılmalıdır. get() ile sorgu compile edilir.

Sorgu get() yazılmadan bırakılırsa $this->db-> ile devam eden sonraki sorgular öncekinin üzerine eklenerek devam eder.

● getResult() nesne dizisi olarak sonuç döndürür.
getRow() nesne olarak tek bir sonuç döndürür.

● Yukarıdaki CustomModel sınıfımızdan devam edelim;

● Tablodaki tüm verileri seçmek için *(SELECT * FROM)*;
```
$this->db->table('tabloAdı')->get()->getResult();
```

● İlk kaydı getirir;
```
$model->where('email', $data['email'])
        ->first();
```

● *WHERE ... AND* sorgusu;
```
$this->db->table('tablo')
        ->where(['sütun' => 'koşul'])
        ->where(['sütun <' => 'koşul'])
        ->where(['sütun !=' => 'koşul'])
        //->where('sütun', 'koşul') ...
        ->get()
        ->getRow(); // veya ->getResult();
```

● *ORDER BY;*
```
$this->db->table('tablo')
        //->where(...) ...
        ->orderBy('sütun', 'DESC veya ASC')
        ->get()
        ->getResult();
```

● *JOIN;*
join() metodu opsiyonel üçüncü paramtre olarak join türünü alır; 'right', 'left', 'inner' (default).
```
$this->db->table('tablo1')
        //->where(...) ...
        ->join('tablo2', 'tablo1.sütun1 = tablo2.sütun2')
        ->get()
        ->getResult();
```

● *LIKE;*
like() metodu opsiyonel üçüncü paramtre olarak join türünü alır;
'both' (%str%, default), 'before' (%str), 'after' (str%).
```
$this->db->table('tablo')
        ->like('sütun', 'koşul')
        //->join( ... )
        ->get()
        ->getResult();
```

● *WHERE ... OR* ve gruplama
*(SELECT * FROM tablo WHERE (koşul1 AND koşul2) OR koşul3);*
```
$this->db->table('tablo')
        ->groupStart() // Grup başlangıcı. ( gibi düşünülebilir.
            ->where(['koşul1' => 'değer1', 'koşul2' => 'değer2'])
        ->groupEnd() // Grup sonu. ) gibi düşünülebilir.
        ->orWhere('koşul3', 'değer3')
```

```
            -//->join( ... )
            ->get()
            ->getResult();
```

● *WHERE IN, LIMIT - OFFSET*
limit() metodu opsiyonel ikinci paramtre olarak offset alır;

```
$this->db->table('tablo')
            // ...
            ->orWhereIn('sütun', $dizi)
            -//->join( ... )
            ->limit(limitDegeri, offsetDegeri)
            ->get()
            ->getResult();
```

● *GROUP BY, COUNT()*
SELECT sütun, COUNT(*) FROM `tablo` GROUP BY sütun;

```
$db = connect_db();
$builder = $db->table('tablo');
$builder->selectCount('sütun'); // or $builder->countAll(); depends on your needs
$builder->groupBy("sütun");
$query = $builder->get();
```

#11 - Form Validation

https://codeigniter4.github.io/userguide/libraries/validation.html

● CodeIgniter'da validation kuralları sadece true veya false döndürür.

● Validation kütüphanesini kullanmak için ilgili controller'ın validation kullanmak istediğimiz metoduna;

```
helper(['helperAdı', 'helperAdı2']);
```

Örnek:

```
helper(['form']);
```

● Validation kurallarını oluşturmak için;

```
$rules = [
        'formInputName' => 'rule1|rul2[param]',
        // Hataları özelleştirmek için;
        'formInputName2' => [
                'rules' => 'rule1|rule2',
                'label' => 'Birşeyler',         // Hata verirken input name yerine 'Birşeyler' kullanacak
                'errors' => [                   // Hataları override ediyoruz
                        'rule1' => 'Email alanını boş bırakamazsınız.',
                        // ...
                ]
        ]
];
```

Örnek:

```
$rules = [
        'email' => 'required|valid_email|is_unique[tablo.sütun]',
        'password' => 'required|min_length[8]',
        'password_confirm' => 'matches[inputName]', // inputName ile yanı mı?
        'category' => 'in_list[Student, Teacher]' // Dropbox'tan kabul edilecek değerler
```

```
];
```

● Kurallara uygun mu diye kontrol etmek için;
```
if($this->validate($rules)) { // ... uygunsa ...
```

● Kurallara uygun değilse hata mesajlarını almak için (diğer bilgileri de verir);
```
$data['degisken'] = $this->validator;
```
Veya direk hataları almak için;
$data['errors'] = $this->validator->geterrors();

● View'da hataları listelemek için;
```
$degisken->listErrors()
```

● Formu gönderdikten (submit) sonra alanların eski değerleriyle dolu olması için set_value() (helper(['form']) ile gelir) metodu kullanmalır;
```
<input type="type" name="name" value="<?= set_value('name', 'opsDefaultVal') ?>">
<option <?= set_select('name', $value) ?> value="value"> ... </option>
```

● Eğer form post edildiyse;
```
if($this->request->getMethod() == 'post') { ...
```

● Başka bir metoda yönlendirmek için;
```
return redirect()->to('/controller/metod');
```

● getVar() metodu ile hem post hem get'teki değerler kontrol edilir. Request'den input değerlerini tek tek almak için;
```
$this->request->getVar('inputName')
```

● Input post edilmiş mi?;
```
if($this->request->getPost('inputName') != '') { ...
```

● CodeIgniter'da hazır validation rule'lar olduğu gibi kendimiz de rule yaratabiliriz.

● Özel kurallar yaratmak için app/Validations klasörünü yarattık. Klasör içinde CustomRules.php dosyamızı oluşturduk ve kuralı tanımladık (isimler önemsiz);
```
namespace App\Validations;
class CustomRules {
	function custom_rule(string $str, string &$error = null) : bool {
		if($str < date('Y-m-d')) {
			return false;
		}
		return true;
	}
}
```

● Oluşturduğumuz kuralı kullanabilmek için app/Config/**Validation.php** dosyasınaki $ruleSets dizisine kledik;
```
public $ruleSets = [
	// ...
	\App\Validations\CustomRules::class,
];
```

● Oluşturduğumuz kuralı, 'errors' 'unu tanımlayarak diğer kurallar gibi kullanabiliriz.

#12 - File Validation & File Upload

https://codeigniter4.github.io/userguide/libraries/uploaded_files.html

● Dosyalara uygulanabilecek validation kuralları;
```
uploaded[inputName]
max_size[inputName, sadeceRakamKb]
ext_in[inputName,uzantı]      Virgülden sonra boşluk yok!
```

● Sadece resimlere özel kurallar;
```
is_image[inputName]
max_dims[inputName,dim,dim]
```

● Dosyaya göz at > aç dedikten sonra dosya instance'ını almak için;
```
$file = $this->request->getFile('inputName');
```

● Dosyayı servera yüklemek için;
```
if($file->isValid() && !$file->hasMoved()) { // CI4 tarafından tavsiye edilen kontrol
        $file->move('./uploads/images'); // Klasör yoksa otomatik olarak oluşturulur.
}
```

● Aynı isimde dosya varsa yani dosya "isim_sayı" formatında kaydedilir.
● Bu yüzden dosya adını almadan önce sunucuya yüklendiğinden emin olmalıyız. Dosya adını almak için;
```
$file->getName()
```

● Dosyayı özel bir isimle yüklemek için;
```
$file->move('./uploads/images', 'isim.' . $file->getExtension());
```

● Rastgele isim vermek için;
```
$file->move('./uploads/images', $file->getRandomName());
```

● Dosya seçilimiş mi dışındaki tüm dosya kuralları çoklu dosya yükleme de geçerli.

● Çoklu yüklemelerde dosya seçilmiş mi kuralı için;
```
uploaded[inputName .0]
```

● Html çoklu dosya kodu;
```
<input type="file" multiple name="name[]">
```

● Çoklu dosyaları sunucuya yüklemek için getFile**s**() metodu kullanılır. ;
```
$files = $this->request->getFiles('inputName'); // Parametresiz de olur.
foreach($files['inputName'] as $file) {
        if($file->isValid() && !$file->hasMoved()) {
                $file->move('./uploads/images/multiple');
        }
}
```

#13 - Image Manipulation

● CodeIgniter resim işleme için GD kütüphanesini kullanıyor.

● Özel helper dosyaları tanımlamak için app/**Helpers** kalsöründe customHelperAdı_helper.php dosyasını oluşturmalı ve içine gerekli yardımcı fonksiyonları yazmalıyız.
* Dosya adının "_helper" ile bitmesi önemli.
● Oluşturulan helper'ın kullanılabilmesi için, kullanmak istediğimiz controller'ın metoduna;
```
helper(['başkaHelperVarsa', 'customHelperAdı']);
```

● Resim manipülasyonu için;

```
$image->withFile('işlemekİstediğimizResminTamYolu')
        ->fit(150, 150, 'center') // Kırp. top|bottom-left|right
        ->save('kaydetmekİstediğimizResminTamYolu');
```
* save() metodu move() metodu gibi otomatik klasör oluşturmaz. Bu yüzden save() metodunu kullanmadan önce klasör var mı diye kontrol etmeli, yoksa oluşturmalıyız.

● fit() metodu dışında flip('yön') ('horizontal' vs.) ve rotate(derece) metodları da var. Daha var...

#14 - Multiple Databases | Working with Multiple Databases

https://codeigniter4.github.io/userguide/database/connecting.html

● .env'de tanımlı veritabanı bilgileri app/Config/**Database.php**'dekileri override eder.

● .env dosyasının kullanılmasının nedenlerinden biri; development ortamından production ortamına geçişte, sadece değişecek veritabanı bilgilerinin .env dosyasında tanımlanması, aynı kalacak olan bilgilerin Database.php dosyasında tanımlanabilmesidir. username gibi...

● Veritabanı bağlantı instance'ını almak için;
```
$db = db_connect();
```

● db_connect() metdouna hiçbir parametre verilmediyse, Database.php dosyasındaki $defaultGroup değişkeninde tanımlı veritabanı kullanılır.

● Defult veritabanı dışında başka veritabanları da kullanacaksak veritabanını Database.php dosyasında tanımlayıp ($default tanımlıdaki gibi) o tanımladığımız değişkenin ismini db_connect() metoduna parametre verek kullanıyoruz.
Ayrıca bu tanımlama .env dosyasında da yapılabilir (default'ta olduğu gibi).

Örnek:
Database.php dosyasında yeni veritabanı tanımı;
```
public $anotherDb = [
        // Veritabanı bilgileri ...
];
```
Controller dosyasında;
```
$db2 = db_connect('anotherDb');
```

● Ayrıca, veritabanı sorguları yapılmadan önce db_connect('db') metoduyla da veritabanı değiştirlebilir fakat birden çok veritabanı kullanımında çok sağlıklı değildir.
Çünkü son ayarlanan veritabanı sonraki geçerli veritabanı da olacaktır ve bu durum farklı veritabanı kullanımı gerektiren tüm işlemlerde yeniden veritbanının tanımlanmasını gerektirebilir.

Örnek:
Özel bir model'in metodunda;
```
$this->db->setDatabase('yeniDb');
$builder = $this->db->table('tablo');
// ...
```

LOGIN ve REGISRATION

CodeIgniter 4 Login & Registration Tutorial

User Registration

● Session flashdata (bir sonraki request için var olur sonra silinir) ayarlamak içi;
```
$session = session();
```

```
$session->setFlashdata('sessionAdı', 'Session İçeriği');
```

● Session değerini almak için;
```
session()->get('degisken')
```

User Login & Session

● Session ayarlamak için;
```
session()->set($data);
```

● URL bilgilerini alabilmek için;
```
$uri = service('uri')
```

● Segmentleri alabilmek için;
```
$uri->getSegment(segmentNo)
```
http://localhost:8080/segment1/segment2/...

Filters, Protecting Routes

● Yetkisiz kullanıclardan route'ları korumak için filter kullanılır. Filtreler aslında request'den önce veya sonra çalışan controller'lardır.

● Özel bir filtre yaratmak için app**/Filters** klasöründe filtre dosyanı oluşturuyoruz.
● İçeriği şu linkten kopyalanabilir;
https://codeigniter.com/user_guide/incoming/filters.html?highlight=controller%20filter
● Sınıf içinde yer alan before() ve after() metodlarının ikisinede kullanmak zorunda değiliz fakat sınıf içinde ikisi de yer almalıdır. Çünkü sınıfımız FilterInterface interface'ini implement eder ve bu interface metodların bulunmasını gerektirir.

● Filtreyi oluşturduktan sonra app/Config/**Filters.php** dosyasındaki $aliases dizisi içerisinde tanımlamalıyız;
```
public $aliases = [
        // ...
        'herhangiBirİsim' => \App\Filters\CustomFiltre::class,
];
```

● Filtreyi kullanmak için Routes.php'de uygulamak istediğimiz route'a paramtre olarak veriyoruz;
```
$routes->get('customUrl', 'Controller::method', ['filter' => 'customFilter']);
```

● Filters.php dosyasındaki $globals dizisindeki 'before' ve 'after' alanlarına tanımlanan filtreler her requeste uygulanır.
Özelleştirilmiş route'ların ilk haline erişimi engellemek için kullanılabilir.

● You are trying to redirect autoroutes (routes that work by default in CI4 - url/controller/method). You can turn them off in Config/Routes.php and than $routes->setAutoRoute(false); so there is no need for UsersCheck.php?