

1 Delo z bazami podatkov

Yii podpira najbolj razširjene relacijske baze podatkov kot so PostgreSQL, MySQL/MariaDB, SQLite, Oracle, MS SQL kot tudi NoSQL MongoDB. Na voljo so tudi razširitve a druge BP.

Za delo z BP je potrebno:

- konfigurirati povezavo na BP
- definirati aktivni zapis (*Active Record*)
- s pomočjo aktivneg zapisa poizvedbo izvesti
- prikazati rezultate

V izbranem SUBP-ju pripravimo bazo podatkov, v našem primeru V PostgreSQL-u z imenom yii, ki bo vsebovala tabelo dijaki. V tabelo vnesemo nekaj zapisov.

```
CREATE TABLE dijaki
(
    iddijaki SERIAL PRIMARY KEY,
    ime VARCHAR(50),
    priimek VARCHAR(50),
    email VARCHAR(50)
);

INSERT INTO dijaki(ime, priimek, email)
VALUES('Peter', 'Klepec', 'peter.klepec@scptuj.si');
INSERT INTO dijaki(ime, priimek, email)
VALUES('Vida', 'Lepa', 'vidal@vidamail.si');
INSERT INTO dijaki(ime, priimek, email)
VALUES('Martin', 'Krpan', 'martin@emajl.com');
INSERT INTO dijaki(ime, priimek, email)
VALUES('Peter', 'Pan', 'pp@emajl.com');
INSERT INTO dijaki(ime, priimek, email)
VALUES('Ana', 'Konda', 'anak@posta.com');
INSERT INTO dijaki(ime, priimek, email)
VALUES('Ali', 'Gator', 'ali@gator.com');
```

1.1 Povezava z BP

Povezavo do BP nastavimo v datoteki config/db.php. Privzeta konfiguracija je nastavljena za delo z MySQL, spremenimo nastavitve.

```
<?php
return [
    'class' => 'yii\db\Connection',
    'dsn' => 'pgsql:host=localhost;port=5432;dbname=yii',
    'username' => 'micka',
    'password' => 'njenogeslo',
    'charset' => 'utf8',
];
```

1.2 Izdelava aktivnega zapisa

Aktivni zapis je razred za predstavitev podatkov iz tabele. Aktivni zapis shranimo v datoteko `models/Dijaki.php`.

```
<?php
namespace app\models;
use yii\db\ActiveRecord;
class Dijaki extends ActiveRecord
{
}
```

Iz imena razreda `Yii` ugotovi ime tabele v bazi, zato ni potrebno dodati nobene kode. Po potrebi lahko z metodo `yii\db\ActiveRecord::tableName()` eksplicitno določimo ime tabele, ki jo naj `Yii` uporabi.

1.3 Izdelava akcije

Izdelamo nov kontroler, da bi ločili med akcijami za delo z BP od od tistih za strani. Kontroler shranimo v `controllers/DijakiController.php`. V njem kreiramo akcijo z imenom `index`.

```
<?php
namespace app\controllers;

use yii\web\Controller;
use yii\data\Pagination;
use app\models\Dijaki;

class DijakiController extends Controller
{
    public function actionIndex()
    {
        $query = Dijaki::find();
        $pagination = new Pagination([
            'defaultPageSize' => 5,
            'totalCount' => $query->count(),
        ]);
        $dijak = $query->orderBy('priimek')
            ->offset($pagination->offset)
            ->limit($pagination->limit)
            ->all();
        return $this->render('index', [
            'dijak' => $dijak,
            'pagination' => $pagination,
        ]);
    }
}
```

Akcija `index` kliče metodo `Dijaki::find()`, ki izvede povpraševanje v BP in posreduje podatke. Povprašnje je oštevilčeno z objektom `yii\data\Pagination`, ki ima dva namena:

- omeji število hkrati vrnjenih zapisov (v našem primeru 5)

- prikaže paginator (pager), seznam gumbov za prehode med stranmi

1.4 Izdelava pogleda

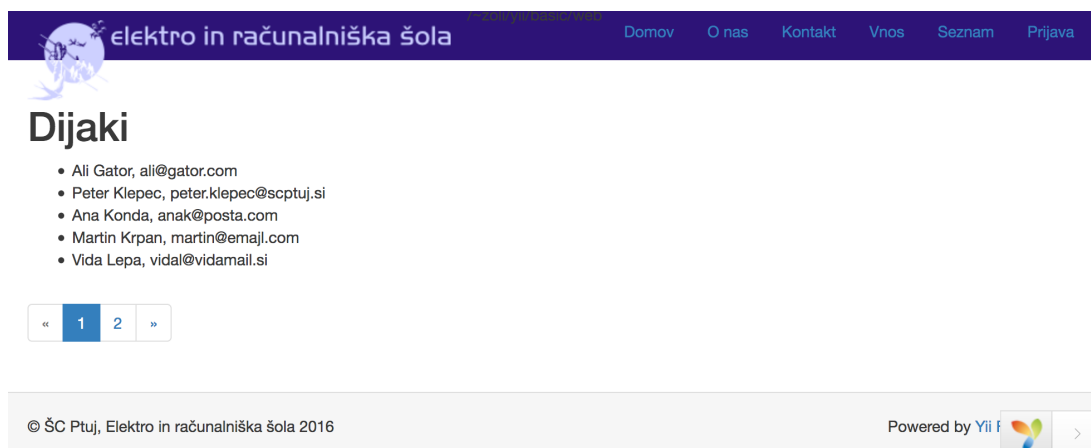
V direktoriju views kreiramo poddirektorij dijaki za poglede s podatki iz BP. Z datoteko views/dijaki/index.php prikažemo podatke iz BP in paginator.

```
<?php
    use yii\helpers\Html;
    use yii\widgets\LinkPager;
?>
<h1>Dijaki</h1>
<ul>
<?php foreach ($dijak as $dijaki): ?>
    <li>
        <?= Html::encode("{ $dijaki->ime} { $dijaki->priimek}") ?>,
        <?= $dijaki->email ?>
    </li>
<?php endforeach; ?>
</ul>
<?= LinkPager::widget(['pagination' => $pagination]) ?>
```

Aplikacijo preizkusimo, v brskalnik vnosemo URL

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

oz. dodamo element v navigacijsko vrstico.



Slika 1: Seznam dijakov iz BP

1.5 Generiranje kode z Gii

Gii je modul, ki omogoča avtomatsko generiranje kode za implementiranje pogostih lastnosti spletnih aplikacij. Generiranje kode z Gii poteka v naslednjih korakih:

- omogočitev Gii v aplikaciji

- generiranje aktivnega zapisa
- generiranje kode za CRUD operacije nad tabelo v BP
- prilagoditev kode generirane z Gii

1.5.1 Zagon Gii

Za zagon Gii mora biti aplikacija v razvojnem načinu. To preverimo v konfiguracijski datoteki `config/web.php` in vstopnem skriptu aplikacije `web/index.php`. V konfiguracijski datoteki mora biti vključen Gii. V primeru dela na oddaljenem računalniku, navedemo IP naslove, s katerih lahko dostopamo do Gii.

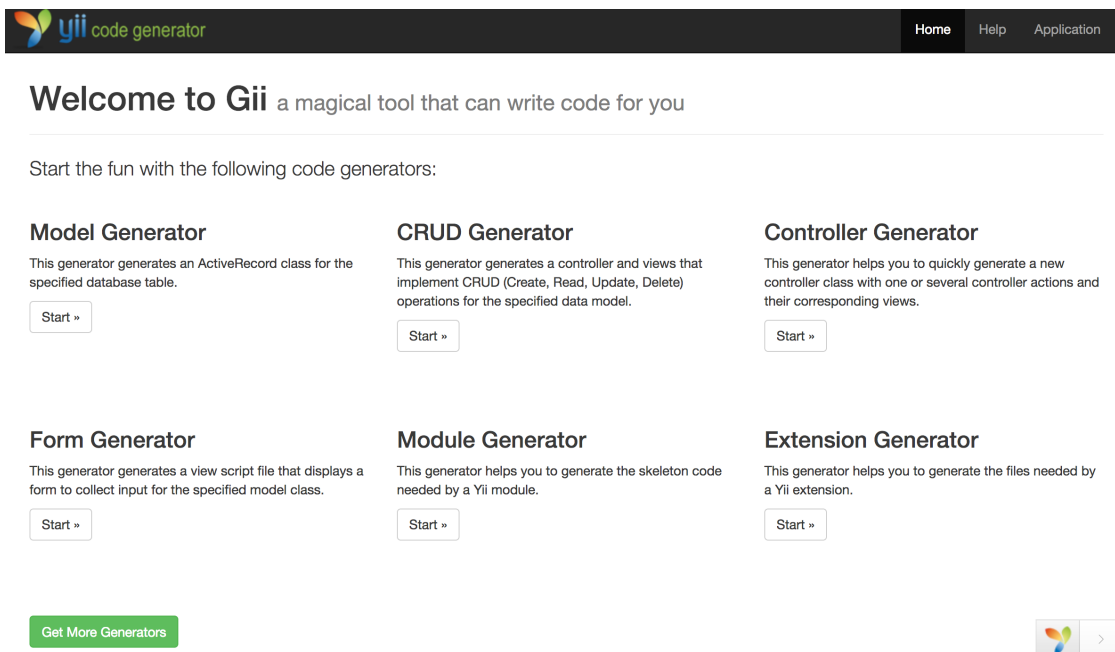
```
if (YII_ENV_DEV) {
    ...
    $config['bootstrap'][] = 'gii';
    $config['modules']['gii'] = [
        'class' => 'yii\gii\Module',
        'allowedIPs' => ['194.249.252.*', '::1'] // IP naslovi, s kat.
                                                // lahko Gii zaženemo
    ];
}
```

Razvojni način v vstopnem skriptu označuje vrstica

```
defined('YII_ENV') or define('YII_ENV', 'dev');
```

Gii zaženem v brskalniku preko URL:

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



Slika 2: Začetna stran Gii

1.5.2 Generiranje aktivnega zapisa

Za generiranje aktivnega zapisa uporabimo *Model Generator*. Na strani, ki se naloži po kliku na gumb <Start>, izpolnimo polji:

- *Table name*: dijaki
- *Model Class*: Dijaki

Nato kliknemo na <Preview>, na dnu strani se prikaže ime datoteke, ki se bo generirala, v našem primeru `models/Dijaki.php`. Ker smo datoteko s tem imenom že ustvarili v poglavju 1.2, označimo potrditveno polje <Overwrite> in kliknemo na <Generate>.

Model Generator

This generator generates an ActiveRecord class for the specified database table.

Table Name

dijaki

Model Class

Dijaki

Namespace

app\models

Base Class

yii\db\ActiveRecord

Database Connection ID

db

☐ Use Table Prefix

Generate Relations

All relations

☐ Generate Labels from DB Comments

☐ Generate ActiveQuery

☐ Enable I18N

☒ Use Schema Name

Code Template

default (/Users/zoli/Sites/yii/basic/vendor/yiisoft/yii2-gii/generators/model/default)

[Preview](#) [Generate](#)

Click on the above **Generate** button to generate the files selected below:

☒ Create ☒ Unchanged ☒ Overwrite

Code File	Action
models/Dijaki.php diff	<input type="checkbox"/> overwrite <input type="checkbox"/>

Slika 3: Izdelava aktivenga zapisa z Gii Model Generator

1.5.3 Generiranje kode za CRUD operacije

CRUD je kratica za operacije CREATE, READ, UPDATE in DELETE, ki so najbolj pogoste operacije nad podatki v spletnih aplikacijah. Zaženemo *CRUD Generator* na začetni strani Gii. Izpolnimo polja:

- *Model Class*: app\models\Dijaki
- *Search Model Class*: app\models\DijakiSearch
- *Controller Class*: app\controllers\DijakiController

Kontroler imamo že od prej, model smo z *Gii Model Generator*-jem posodobili. Kliknemo na <Preview>. Na dnu strani se pojavi seznam datotek, ki jih bo generator ustvaril. Označimo potrditvena polja <Overwrite> za tista, ki že obstajajo in kliknemo <Generate>.


Code File	Action	
controllers/DijakiController.php	overwrite	<input checked="" type="checkbox"/>
models/DijakiSearch.php	create	<input checked="" type="checkbox"/>
views/dijaki/_form.php	create	<input checked="" type="checkbox"/>
views/dijaki/_search.php	create	<input checked="" type="checkbox"/>
views/dijaki/create.php	create	<input checked="" type="checkbox"/>
views/dijaki/index.php	overwrite	<input checked="" type="checkbox"/>
views/dijaki/update.php	create	<input checked="" type="checkbox"/>
views/dijaki/view.php	create	<input checked="" type="checkbox"/>

Slika 4: Gii CRUD Generator

Stran preizkusimo, v brskalnik vnesemo URL:

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

Stran, ki jo je Gii pripravil, omogoča poleg operacij CRUD nad podatki (gumbi v zadnjem stolpcu) še pošiljanje el. pošte, urejanje zapisov s klikom na ime stolpca in filtriranje podatkov (polja med naslovno vrstico in vsebino tabele).


elektro in računalniška šola

[Domov](#)
[O nas](#)
[Kontakt](#)
[Vnos](#)
[Seznam](#)
[Prijava](#)

[Domov](#) / [Dijaki](#)

Create Dijaki

© ŠC Ptuj, Elektro in računalniška šola 2016

Powered by [Yii](#)

Slika 5: Stran generirana z Gii

1.5.4 Prilagoditev pogleda

Stran, ki smo jo izdelali s pomočjo Gii, želimo še prilagoditi lastnim potrebam, in sicer:

- moti nas napis na gumbu v anleščini
- id-jev dijakov ne želimo prikazati
- ne dovolimo brisanja dijakov

Spremembe opravimo v datoteki `views/dijaki/index.php`.

```

...
<div class="dijaki-index">
...
    <?= Html::a('Nov dijak', ['create'], ['class' => 'btn btn-
        success']) ?>
</p>
    <?= GridView::widget([
        'dataProvider' => $dataProvider,
        'filterModel' => $searchModel,
        'columns' => [
            ['class' => 'yii\grid\SerialColumn'],
//            'iddijaki', // skrijemo stolpec iddijaki
            'ime',
            'priimek',
            'email:email',
            ['class' => 'yii\grid\ActionColumn', // gumb delete skrijemo
                'visibleButtons' => ['delete' =>
                    function ($model, $key, $index) {return false;}}]
        ],
    ]); ?>

```

</div>

Dodamo še paginator na dno tabele za navigacijo po rezultatih, datoteka `DijakiSearch.php`:

```
...
public function search($params)
{
    $query = Dijaki::find();
    $dataProvider = new ActiveDataProvider([
        'query' => $query,
        'pagination' => ['pageSize' => 5,]
    ]);
    ...
}
```

Po zgoraj opravljenih spremembah bo stran s seznamom dijakov kot ga prikazuje Slika 6.

elektro in računalniška šola Domov O nas Kontakt Vnos Seznam Prijava

Domov / Dijaki

Dijaki

Nov dijak

Showing 1-5 of 6 items.

#	Ime	Priimek	Email	
1	Peter	Klepec	peter.klepec@scptuj.si	
2	Vida	Lepa	vidal@vidamail.si	
3	Peter	Pan	pp@emajl.com	
4	Martin	Krpan	martin@emajl.com	
5	Ali	Gator	ali@gator.com	

« 1 2 »

© ŠC Ptuj, Elektro in računalniška šola 2017 Powered by Yii Framework

Slika 6: Stran po prilagoditvi

Ne podoben način lahko prilagodimo tudi ostale poglede v direktoriji `views/dijaki`, ki jih je generiral Gii.

1.6 Objekti za dostop do baze podatkov

Objekti za dostop do baze podatkov (*Database Access Objects - DAO*) so programski vmesnik (*API*) za dostop do relacijskih baz podatkov. So osnova za bolj napredne metode kot v prejšnjem poglavju opisani aktivni zapis (*Active Record*) in graditelj povpraševalnih stavkov (*Query Builder*), ki ga bom spoznali kasneje.

S pomočjo DAO lahko izvedemo poljubni SQL stavek. Izvedba opravimo v treh korakih:

1. kreiramo `yii\db\Command`-o z čistim SQL-om
2. podamo parametre (opcijsko)
3. kličemo eno od metod `queryXyz()` za izvedbo `yii\db\Command`-e:

- **queryAll()** – vrne množico vrstic, vsaka vrstica je asociativno polje z indeksom imena atributa, oz. prazno polje, če ne najde zapisov

```
$sql = "SELECT * FROM dijaki;";  
$dijaki = Yii::$app->db->createCommand($sql)->queryAll();
```

- **queryOne()** – vrne eno vrstico (prvo), `false` če ne najde zapisov

```
$sql = "SELECT * FROM dijaki WHERE id = 13;";  
$dijak = Yii::$app->db->createCommand($sql)->queryOne();
```

- **queryColumn()** – vrne stolpec, prazno polje, če ne najde zapisov

```
$sql = "SELECT ime FROM dijaki;";  
$imena = Yii::$app->db->createCommand($sql)->queryColumn();
```

- **queryScalar()** – vrne število, `false`, če ni rezultata

```
$sql = "SELECT count(*) FROM dijaki;";  
$stevalo = Yii::$app->db->createCommand($sql)->queryScalar();
```

Zaradi zaščite pred SQL injekcijami je parametre v SQL povpraševalni stavek priporočljivo podati z metodami:

- **bindValue()** – podamo posamezni paramater

```
$sql = "SELECT ime, priimek FROM dijaki WHERE ime LIKE :ime AND  
priimek LIKE :priimek;";  
$data = Yii::$app->db->createCommand($sql)  
->bindValue(':ime' => 'P%')  
->bindValue(':priimek' => 'P%')  
->queryAll();
```

- **bindValues()** – podamo več parametrov hkrati

```
$sql = "SELECT ime, priimek FROM dijaki WHERE ime LIKE :ime AND  
priimek LIKE :priimek;";  
$params = [':ime' => 'P%', ':priimek' => 'P%'];  
$data = Yii::$app->db->createCommand($sql)  
->bindValues($params)  
->queryAll();
```

- **bindParam()** – podobno kot `bindValue()`, omogoča referenco na parameter

```
$sql = "SELECT ime, priimek FROM dijaki WHERE id = :id";  
$comm = Yii::$app->db->createCommand($sql)->bindParam(':id', $id);  
  
$id = 1;  
$dijak1 = $comm->queryOne();  
  
$id = 2;  
$dijak2 = $comm->queryOne();
```

Za izvedbo nepovpraševalnih SQL stavkov uporabimo metodo `execute()`:

```
$sql = "INSERT INTO dijaki(ime, priimek, email)
VALUES('Micky', 'Mouse', 'micky@mouse.com');
Yii::$app->db->createCommand($sql)->execute();
```

Za stavke `INSERT`, `UPDATE` in `DELETE` imamo na voljo posebne metode `insert()`, `update()` in `delete()`:

```
Yii::$app->db->createCommand()->insert('dijaki',
['ime' => 'Micky', 'priimek' => 'Mouse',
'email => 'micky@mouse.com',])->execute();
```

Uporabo *DAO* demonstriramo na aplikaciji izdelani z Gii v poglavju 1.5. Iz BP želimo poiskati vse dijake, katerih ime in priimek se začneta na črko P.

V model `DijakiSearch.php` dodamo metodo s poizvedovanjem:

```
...
public static function getDijaki()
{
    $sql = "SELECT ime, priimek FROM dijaki WHERE ime LIKE :ime AND
        priimek LIKE :priimek;";
    $params = [':ime' => 'P%', ':priimek' => 'P%'];
    $data = Yii::$app->db->createCommand($sql)->bindValues($params)
        ->queryAll();
    return $data;
}
...
```

V kontrolerju `DijakiController.php` modificiramo metodo `actionIndex()` – podatke bomo prikazali v pogledu `index.php`.

```
...
public function actionIndex()
{
    ...
    $searchModel = new DijakiSearch();

    $data = $searchModel->getDijaki();
    return $this->render('index' , array(data=>$data, ));
}
...
```

Pogled `index.php` prikažemo podatke kot običajno besedilo, saj so podatki pridobljeni z metodo `getDijaki()` v obliki množici polj in ne kot objekt razreda `ActiveDataProvider()`.

```
...
foreach ($data as $row)
{
    echo $row['ime'] . ' ' . $row['priimek'] . "<br />";
}
...
```

1.7 Graditelj povpraševalnih stavkov

Graditelj povpraševalnih stavkov (*Query Builder*) omogoča bolj izdelavo bolj čitljive kode in

generiranje varnejša SQL povpraševanja umerjavi z izvajanjem surove SQL kode. S *Query Builder*-jem sestavimo povpraševalni stavek s posameznimi komponentami stavka SELECT nato povpraševanje izvedemo, npr. z metodo `all()`.

Metode za izgradnjo povpraševalnega stavka so:

- `select()` – predstavlja SELECT del povpraševalnega stavka, parametri so atributi tabele
- `from()` – predstavlja FROM del povpraševalnega stavka, parametri so imena tabel
- `where()` – predstavlja WHERE del povpraševalnega stavka, parameter je pogoj, dodatno lahko uporabimo `andWhere()` in `orWhere()` za sestavljen pogoj
- `orderBy()`
- `limit()` itd

S pomočjo *Query Builder*-ja izvedemo funkcionalno enako povpraševanje kot z aktivnim zapisom v poglavju 1.5.

V model `DijakiSearch.php` dodamo metodo `getDijaki()` s poizvedovanjem, rezultat je enak metodi `search($params)`, ki ga je generiral Gii:

```
...
public static function getDijaki()
{
    $query = new \yii\db\Query;
    $query->select('ime, priimek, email')->from('dijaki');
    $dataProvider = new ActiveDataProvider([
        'query' => $query,
        'pagination' => [
            'pageSize' => 5,
        ],
    ]);
    return $dataProvider;
}
...
```

V kontrolerju `DijakiController.php` modificiramo metodo `actionIndex()`:

```
...
public function actionIndex()
{
    $searchModel = new DijakiSearch();

    // nadomestimo prvotno iskalno metodo
    // $dataProvider = $searchModel->search(Yii::$app->request
    // ->queryParams);

    $dataProvider = $searchModel->getDijaki();

    return $this->render('index', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
    ]);
}
...
```

Podatke prikažemo v pogledu `index.php` enako kot v poglavju 1.5, slika 6.

2 Literatura in viri:

- [1] The Definitve Guide to Yii 2.0 - <http://www.yiiframework.com/doc-2.0/guide-start-workflow.html>
- [2] 10 PHP Frameworks For Developers – Best Of <http://www.hongkiat.com/blog/best-php-frameworks/>
- [3] When & Why to Use PHP Framework? – <http://www.templatemonster.com/blog/php-frameworks-when-and-why-to-use-them/>
- [4] How to choose a PHP framework, <https://opensource.com/business/16/6/which-php-framework-right-you>