Composer is the main feature of MODX 3

First alpha version was released on December 2019.

Stable version was released in March 2022.

MODX 3 uses Composer under the hood, but we do not.

How it works right now

Every extra can provide `bootstrap.php` with its own dependencies.

MODX will load **all** this files on every request.

```php
public function initialize($contextKey= 'web', $options = null) {
    //...
    $this->_initNamespaces();
    // ...
}

protected function _initNamespaces() {
    // ...
    foreach ($namespaces as $namespace) {
        if (is_readable($namespace['path'] . 'bootstrap.php')) {
            require $namespace['path'] . 'bootstrap.php';
        }
    }
}
```

Typical bootstrap.php

For example, **FormIt** extra:

```php
<?php

require_once __DIR__ . '/vendor/autoload.php';

// Add your classes to modx's autoloader
$modx->addPackage('Sterc\FormIt\Model', $namespace['path'] . 'src/', null, 'Sterc\\FormIt\\');

// Register base class in the service container
$modx->services->add('formit', function() use ($modx) {
    return new \Sterc\FormIt($modx);
});
```

The problem

No one knows what dependencies can be added!

Some components can use the same dependencies with different versions.

We do not use Composer to resolve potential conflicts, but that's what it was designed for.

The solution

We need to start using the core `composer.json` as the only source of dependencies.

Which means... **All extras must be installed with Composer.**

It is a long way, but the only way...

Pros

- Proper versioning and releases.
- Using Packagist.org infrastructure.
- All dependency conficts must be resolved before installation!
- Useful skills for developers.

Contras

- Using CLI instead of MODX package manager.

Bonus

- Ability to rewrite all needed extras from the scratch.

MMX stands for Modern MODX

The new type of composer-only extras.

Start using Composer with MODX 3

```
cd /to/modx/root/
wget https://raw.githubusercontent.com/modxcms/revolution/v3.0.4-pl/composer.json
```

Install new component from https://github.com/bezumkin/mmx-forms

```
composer require mmx/forms
composer exec mmx-forms install
```

Demo - https://youtu.be/RjzuqWIkhPc

How to develop?

https://github.com/bezumkin/mmx-app

```
git clone https://github.com/bezumkin/mmx-app.git
cd ./mmx-app

cp .env.dist .env
./run-rename.py any-new-name

docker compose up --build -d

./run-install.sh
```

Demo - https://youtu.be/lwDdZYYBbPw

Features

Nice manager application:

- @vesp/mmx-frontend library
- Vue 3, Bootstrap 5, Vite with HMR

Powerfull backend with Vesp:

- Eloquent models
- Phinx migrations
- Slim 4 routes with real REST API
- Built-in CLI

Install and remove

Built-in CLI scripts do all the work:

```php
public function run(InputInterface $input, OutputInterface $output): void
{
    $srcPath = MODX_CORE_PATH . 'vendor/' . preg_replace('#-#', '/', App::NAMESPACE, 1);
    $corePath = MODX_CORE_PATH . 'components/' . App::NAMESPACE;
    $assetsPath = MODX_ASSETS_PATH . 'components/' . App::NAMESPACE;

    if (!is_dir($corePath)) {
        symlink($srcPath . '/core', $corePath);
        $output->writeln('<info>Created symlink for "core"</info>');
    }

    if (!is_dir($assetsPath)) {
        symlink($srcPath . '/assets/dist', $assetsPath);
        $output->writeln('<info>Created symlink for "assets"</info>');
    }

    if (!Namespaces::query()->find(App::NAMESPACE)) {
        $namespace = new Namespaces();
        $namespace->name = App::NAMESPACE;
        $namespace->path = '{core_path}components/' . App::NAMESPACE . '/';
        $namespace->assets_path = '{assets_path}components/' . App::NAMESPACE . '/';
        $namespace->save();
        $output->writeln('<info>Created namespace "' . $namespace->name . '"</info>');
    }

    // ...
}
```

Handle requests

One plugin to handle all events:

```
$modx->services->get('mmxApp')?->handleEvent($modx->event);
```

The method:

```
public function handleEvent(?modSystemEvent $event): void
{
    if ($event->name === 'OnManagerPageInit') {
        // Load manager page
        if ($event->params['namespace'] === $this::NAMESPACE) {
            class_alias(Controllers\Modx\Home::class, '\MODX\Revolution\Controllers\Home');
        }

    } elseif ($event->name === 'OnHandleRequest') {
        // Handle API requests
        if (str_starts_with($_SERVER['REQUEST_URI'], '/' . $this::NAMESPACE)) {
            $this->run();
            exit();
        }
    }
}
```

Register assets in production mode

```php
$script = 'src/mgr.ts';

$baseUrl = MODX_ASSETS_URL . 'components/' . self::NAMESPACE . '/';
$manifest = MODX_ASSETS_PATH . 'components/' . self::NAMESPACE . '/manifest.json';

// Try to read manifest
if (file_exists($manifest) && $files = json_decode(file_get_contents($manifest), true)) {
    $assets = [];
    if (!empty($files[$script])) {
        $file = $files[$script];
        $assets[] = $baseUrl . $file['file'];
        foreach ($file['css'] as $css) {
            $assets[] = $baseUrl . $css;
        }
    }

    // Register files
    foreach ($assets as $file) {
        if (str_ends_with($file, '.js')) {
            $instance->addHtml('<script type="module" src="' . $file . '"></script>');
        } else {
            $instance->addCss($file);
        }
    }
}
```

Register assets in development mode

```php
$port = getenv('NODE_DEV_PORT') ?: '9090';
$connection = @fsockopen('node', $port);

if (@is_resource($connection)) {
    $server = explode(':', MODX_HTTP_HOST);

    $baseUrl = MODX_ASSETS_URL . 'components/' . self::NAMESPACE . '/';
    $vite = MODX_URL_SCHEME . $server[0] . ':' . $port . $baseUrl;

    $instance->addHtml('<script type="module" src="' . $vite . '@vite/client"></script>');
    $instance->addHtml('<script type="module" src="' . $vite . 'src/mgr.ts"></script>');
}
```

This way you will have hot modules reload (HMR)!

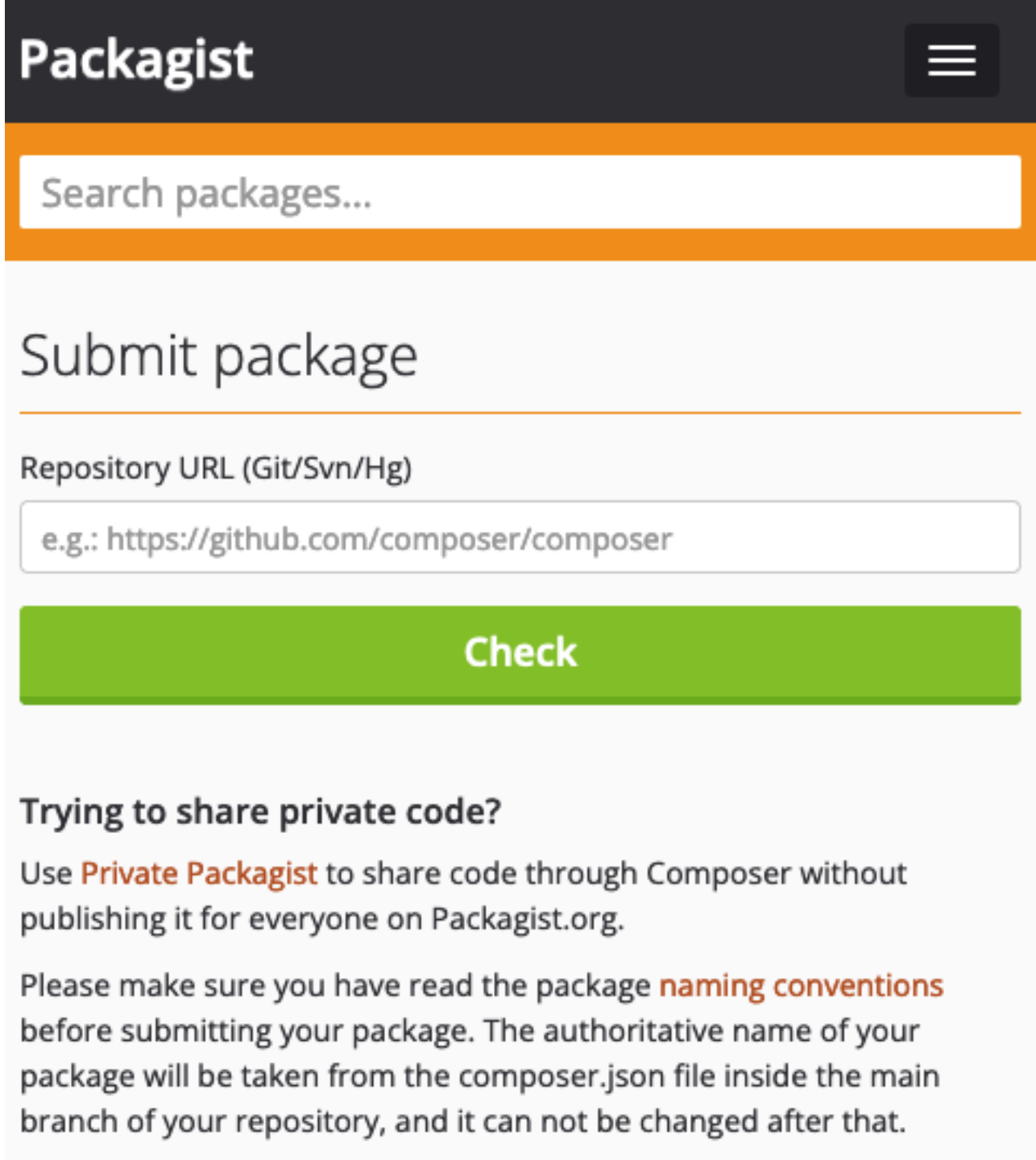
How to publish

1. Build frontend

```
./run-build.sh
```

2. Export component to Github.
3. Submit component to Packagist.
4. Good job!

You can add new public repository as submodule for the dev version.



Thank you!