DEVELOPING FOR WORDPRESS

DO'S AND DONT'S

Hashtag for today: **#WPKonferenz**

Created by **Silvan Hagen** / **@neverything**

WHAT'S ON THE MENU TODAY?

DEVELOPING THEMES

- Where to start?
- How to use assets
- Child Theme or not?
- `Code:` Theme or Plugin?
- Ressources

DEVELOPING PLUGINS

- Where to start?
- Meet the APIs
- The Good and the Bad
- Ressources

THEMES

WHERE TO START?

FROM SCRATCH?

Nah, there are too many solid boilerplates!

WITH A THEME FRAMEWORK OR BLANK THEME?

There are some fantastic WordPress theme frameworks and blank themes!

FREE YEAH!

- **Underscores (_S)**
- **Thematic**
- **Hybrid** (paid option available)
- **UpThemes Framework**
- **Roots**
- **Bones**
- **required+ Foundation** (shameless advertisting)

PAID

- Genesis
- PageLines
- Headway

SO SHOULD I USE A THEME FRAMEWORK?

Yes, but it depends. Look at them individually and decide for yourself and the project ahead.

I usually strive for base themes or frameworks that are very close to WordPress using the **original template structure**, not bloated with options, and so on.

HOW TO USE ASSETS?

Stylesheets, JavaScripts and other external ressources in your theme.

MEET ACTION AND FILTER HOOKS

> *“ Hooks are the backbone of WordPress. They enable developers to hook into the WordPress workflow to change how it works without directly modifying the core code. ”*

Professional WordPress Development - Chapter 3

EXAMPLE OF AN ACTION AND A FILTER HOOK

Example of an **action hook**:

```
function req_email_friends() {
        wp_mail(...);
}
add_action ( 'wp_title', 'req_email_friends' );
```

Example of a **filter hook**:

```
function req_add_heart_to_title( $title, $sep ) {
        $heart = '♥';
        // Append the heart to the title
        $title .= $sep . ' ' . $heart;
        return $title;
}
add_filter( 'wp_title', 'req_add_heart_to_title', 10, 2 );
```

YOU COULD DO - BUT SHOULDN'T

```
function req_my_scripts() {
        echo '<script src="<?php echo get_stylesheet_directory_uri(); ?
>/javascripts/application.js"></script>';
}
add_action ( 'wp_head', 'req_my_scripts' );
```

It's a little better than putting everything in the `header.php`, but how about letting WordPress know about the assets in your theme?

YOU SHOULD DO

Example from Underscores:

```
/**
 * Enqueue scripts and styles
 */
function _s_scripts() {
        wp_enqueue_style( 'style', get_stylesheet_uri() );

        wp_enqueue_script(
                'small-menu', // ID
                get_template_directory_uri() . '/js/small-menu.js',
                array( 'jquery' ), // dependencies
                '20120206', // version
                true // in footer
        );
}
add_action( 'wp_enqueue_scripts', '_s_scripts' );
```

This will still print your themes stylesheet in the header through `wp_head()` and add the `small-menu.js` to the footer using `wp_footer()`.

CHILD THEME OR NOT?

Inherit the good (and the bad) stuff from your parents can be handy!

WHAT THE HECK IS A CHILD THEME?

> *“ A WordPress child theme is a theme that inherits the functionality of another theme, called the parent theme, and allows you to modify, or add to, the functionality of that parent theme. ”*

The Codex on Child Themes

CHILD THEMES IN A NUTSHELL

```
/wp-content
        /themes
                /your-parent-theme      // your example parent theme
                /your-child-theme       // name doesn't matter here
                        style.css       // required to make it work
```

In your child themes `style.css`

```
/*
Theme Name: Your Child Theme
Description: Child theme for the Your Parent Theme theme
Author: Your name here
Template: your-parent-theme
*/
@import url("../your-parent-theme/style.css");
```

This is all it takes to create a child theme. So far it does nothing different than the parent.

THEME OR PLUGIN?

Where should you put your custom code? An ongoing discussion in the community.

DON’T BLOAT YOUR THEME WITH SHORTCODES!

... and don't believe in theme authors selling you `[shortcodes]` as theme features.

DEVELOP UNIVERSAL SHORTCODE PLUGINS!

A good example is the
Grid Columns Plugin by Justin Tadlock

WHAT ABOUT CUSTOM POST TYPES?

Custom Post Types are WordPress one-fits-all (most) model to store content in almost any form you like.

So **it's up to you** wether you add the custom post type to your theme or create a plugin.

RESSOURCES

With a community this big, consider yourself lucky!

CREATE A THEME

- The ThemeShaper WordPress Theme Tutorial: 2nd Edition
- Create a child theme with TwentyTwelve

DEBUG YOUR THEME

- 5 Ways to Debug WordPress
- Debug Bar Plugin
- Developer Plugin
- Log Deprecated Notices

TEST YOUR THEME

- Official Theme Unit Test
- Theme-Check Plugin

PLUGINS

WHERE TO START?

FROM SCRATCH?

Nope, there is the fantastic **WordPress Plugin Boilerplate** and even a **code generator** for the lazy ones.

USING THE WORDPRESS PLUGIN BOILERPLATE

```
class DemoPlugin {
    public function __construct() {
        load_plugin_textdomain( 'demo-plugin', false,
            dirname( plugin_basename( __FILE__ ) ) . '/lang' );

        add_action( 'wp_enqueue_scripts',
            array( $this, 'register_plugin_styles' ) );
        add_action( 'wp_enqueue_scripts',
            array( $this, 'register_plugin_scripts' ) );
        add_filter( 'the_content',
            array( $this, 'append_post_notification' ) );
    }
}
```

This example is from **WP Tuts+**.

CREATE YOUR OWN?!

After you wrote a few plugins yourself, you might feel the need to create your own boilerplate.

MEET THE APIS

WordPress currently **lists 16 different APIs** available for you to work with.

PLUGIN API

WordPress **Plugin API** is the **action and filter hooks system**, that allows you to hook in where your plugin has to.

```
// Create a new filter hook
$my_args = apply_filters( 'req_args_filter', $args );

// Create your own action hook
do_action( 'req_header_action' );
```

PLUGIN API

Now people can filter your `$my_args` and attach an action to `req_header_action`:

```
// Add a filter to $my_args
function my_custom_args_filter( $args ) {
        // do something with the data
        return $args;
}
add_filter( 'req_args_filter', 'my_custom_args_filter' );

// Add an action to req_header_action
function my_custom_header_action() {
        // Do whatever you like ;-)
        echo 'Hello my friend';
}
add_action( 'req_header_action', 'my_custom_header_action' );
```

PLUGIN API

Your plugin might leave data as options or you need custom tables, please make use of the **activation/deactivation and uninstall hooks** to add or remove data.

- `register_activation_hook()` run code on plugin activation
- `register_deactivation_hook()` run code on plugin deactivation
- `register_uninstall_hook()` or `uninstall.php` here you actually delete whatever your plugin created

OPTIONS API

Need to save global plugin options? Use the **Options API**.

```php
// Store some data
$my_option = array( 'state' => 'yellow', 'version' => '0.1.0' );
update_option( 'my_plugin_option', $my_option );

// Get the data back
$options        = get_option( 'my_plugin_option' );
$state          = $options['state'];
$version        = $options['version'];
```

SETTINGS API

> *“ Dealing with user inputs introduces new constraints in the option process: You need to design a user interface, monitor form submissions, handle security checks, and validate user inputs. To easily manage these common tasks, WordPress wraps the option functions into a comprehensive Settings API. ”*

Professional WordPress Development - Chapter 7

SETTINGS API

The **Settings API** offers you to register groups of settings and settings fields. It also helps creating the form and handles the errors for you.

HTTP API

The **HTTP API** is an easy and standardized API to fetch remote data.

```
$request = wp_remote_get( 'http://example.com' );
$body = wp_remote_retrieve_body( $request );
// $body contains the actual page content returned by the server
```

TRANSIENTS API

With the **Transients API** you can store data that expires.

```
// Get any existing copy of our transient data
if ( false === ( $custom_query = get_transient( 'special_query' ) ) ) {
    // It wasn't there, so regenerate the data and save the transient
     $custom_query = new WP_Query( 'cat=5&order=random&tag=tech' );
     set_transient( 'special_query', $custom_query, 12 * HOUR_IN_SECOND
S );
}
// Use the data like you would have normally...
```

THE GOOD AND THE BAD

Some simple rules for plugin development and how to find good plugins.

THE BAD

If you do any of these things you gonna have a bad time.

- Includes a custom version of jQuery
- Loads JS and CSS on all requests and admin pages
- Not ready for translation or doing it wrong
- Direct DB access without `$wpdb->prefix`
- Want some more? Visit **Crappy Code**

THE GOOD

It's not that hard to be the good guy, test your stuff and respect others.

- Respect the global namespace
- Prefer API methods over direct access
- Add custom action and filter hooks where appropriate
- Stay close to WordPress styling in the backend
- Only use custom DB tables when needed

RESSOURCES

Ready to write better plugins? Here are some inputs!

CREATE A PLUGIN

- The Codex on Writing a Plugin
- Develop Plugins: Object-Oriented Programming
- Develop Plugins: Functional Programming

INTERNATIONALIZATION AND TRANSLATIONS

- Internationalization: You're probably doing it wrong
- More Internationalization Fun

WORDPRESS UI

- Integrating With WordPress' UI: The Basics
- WordPress Admin Style Plugin

BOOKS

- Digging into WordPress
- Professional WordPress Plugin Development

FIVE MORE THINGS!

1. The core is your friend, read the WordPress code.
2. Inspire yourself by digging through great plugins.
3. Get involved in the community, we need everyone.
4. Localize your stuff, so people can **easily translate it**.
5. **Go and build something awesome!**

QUESTIONS?

Further questions? Contact me: **silvan@required.ch** / **@neverything**

THANK YOU!

This presentation is available on **GitHub**, **SlideShare** and **our website**.