1.2.1 Recommended Local Dev Setup: LocalWP / Docker Options Conceptually □□

This lesson is about building a **safe, fast, professional development environment** for WordPress animation work with **BricksBuilder, GSAP, and PhpStorm**.

If you want to become *really good* at GSAP on WordPress sites, your setup matters more than most beginners realize. A weak setup causes confusion:

1. You don’t know whether a bug comes from:
    1. your code,
    2. WordPress,
    3. caching,
    4. the host,
    5. or Bricks.
2. You avoid experimenting because breaking a live site feels risky.
3. You waste energy on deployment and environment problems instead of learning animation.

A solid local setup fixes that. □

What you are trying to achieve

Your goal is to create a local WordPress environment where you can:

1. build pages in **BricksBuilder**,
2. write and organize **JavaScript, CSS, and PHP** cleanly,
3. test **GSAP animations** safely,
4. debug quickly in the browser and in PhpStorm,
5. later move your work to staging/live with fewer surprises.

Think of local development as your **animation laboratory** □□

You should be able to:

1. break things without fear,
2. inspect everything,
3. reset quickly,
4. test ideas fast.

The two main local development approaches

For this course, the two main conceptual paths are:

1. **LocalWP**
2. **Docker-based local environment**

Both are valid.
Both can be professional.
But they serve slightly different types of learners and workflows.

Option 1: LocalWP

What LocalWP is

LocalWP is a desktop application designed to make local WordPress development easy.

It handles things like:

1. PHP,
2. MySQL or MariaDB,
3. web server setup,
4. SSL,
5. site creation,
6. local domains.

Instead of manually configuring all those layers, you click a few buttons and get a working WordPress site.

For a beginner, this is often the best choice because it reduces setup friction dramatically.

Why LocalWP is excellent for this course

For learning GSAP with WordPress and Bricks, LocalWP is great because:

1. **Fast setup**
    1. You can create a new site in minutes.
    2. You spend your time learning animation, not server administration.
2. **Good for experimentation**
    1. You can spin up test sites for specific lessons.
    2. You can destroy and recreate environments with low risk.
3. **WordPress-friendly**
    1. It is built around WordPress workflows.
    2. It feels less abstract than Docker for newcomers.
4. **Easy SSL and local domains**
    1. Useful when testing scripts and realistic browser behavior.
    2. Makes your local environment feel closer to a real site.
5. **Lower cognitive load**
    1. As a beginner, you already need to learn GSAP, JavaScript, CSS, Bricks, and WP structure.
    2. LocalWP avoids adding “container orchestration” to that list too early.

When LocalWP is the right choice for you

Choose **LocalWP first** if:

1. you are still learning WordPress internals,
2. you want to focus on **GSAP + Bricks** rather than DevOps,
3. you want the easiest path to a working local site,
4. you work mostly alone on your machine,
5. you value convenience over full environment portability.

For most students starting this course, **this is my recommendation**

Potential downsides of LocalWP

LocalWP is convenient, but it is not perfect.

1. **Less explicit infrastructure knowledge**
    1. A lot is handled for you.
    2. That’s good at first, but it can hide how the environment actually works.
2. **Machine-specific quirks**

    1. Your setup may behave slightly differently from another developer’s.
    2. This matters more in teams.
3. **Less “infrastructure as code”**
    1. Docker setups are often more reproducible.
    2. With LocalWP, your environment is more GUI-managed.
4. **Advanced customization may be less elegant**
    1. You *can* customize things,
    2. but Docker often gives more systematic control.

These are not deal-breakers. They only matter more as your workflow becomes more advanced.

Option 2: Docker

What Docker is

Docker lets you define your development environment in containers.

A container is a packaged runtime environment for part of your app, such as:

1. PHP,
2. database,
3. web server,
4. mail tools,
5. node tooling.

Instead of saying “it works on my machine,” Docker tries to make the machine behavior more predictable by defining the environment in configuration files.

For WordPress, this often means you have services such as:

1. `wordpress` or `php`,
2. `nginx` or `apache`,
3. `mysql` or `mariadb`,
4. optionally `phpmyadmin`,
5. optionally `mailhog`,
6. optionally `node`.

Why Docker is powerful

Docker is attractive because:

1. **Reproducibility**
    1. Team members can use the same environment definition.
    2. Fewer “works for me” problems.
2. **Version control of environment**
    1. Your environment config can live in the project.
    2. That means setup becomes part of the codebase.
3. **Closer to professional engineering workflows**
    1. Useful if you want to work in teams or agencies.
    2. Useful if you deploy to containerized infrastructure.
4. **More explicit setup**
    1. You become more aware of the layers involved.
    2. That deeper knowledge can help debugging.

Why Docker may be harder at first

For a beginner, Docker adds complexity:

1. You may need to understand:
    1. containers,
    2. images,
    3. ports,
    4. volumes,
    5. service names,
    6. networking.
2. When something breaks, the failure may come from:
    1. WordPress,
    2. PHP,
    3. file permissions,
    4. networking,
    5. container config.
3. That can distract from the main goal of this course:
    1. learning **GSAP deeply**,
    2. inside **WordPress and Bricks**.

So Docker is excellent—but often better as a **phase-2 setup**, not necessarily day-1.

My recommendation for *this* course

Here is the practical recommendation:

1. **Start with LocalWP**
    1. It will get you productive fastest.
    2. It reduces noise while you learn core concepts.
2. **Understand Docker conceptually**
    1. So you know what more advanced teams may use.
    2. So you can migrate later if needed.
3. **Move to Docker later only if one of these becomes true**
    1. you work in a team,
    2. you need reproducible onboarding,
    3. you want environment config in version control,
    4. you become comfortable enough that the added complexity is worth it.

So: **LocalWP now, Docker awareness from the start** □□

The professional mindset: environment goals

No matter which tool you choose, a professional local environment should give you these capabilities:

1. **Isolation**
    1. Each project should be able to run without polluting other projects.
    2. Different PHP or plugin setups should not constantly clash.
2. **Repeatability**
    1. You should be able to recreate the setup.
    2. “I forgot what I installed” is a bad sign.
3. **Visibility**
    1. You should be able to inspect logs, files, network requests, and script loading.
    2. Animation debugging depends on visibility.
4. **Speed**
    1. Slow feedback kills experimentation.
    2. GSAP learning requires many tiny test iterations.
5. **Safety**
    1. You should be free to test destructive changes locally.
    2. Never use a live site as your primary learning playground.

A great starter architecture for your learning setup

Here is the setup I recommend for you as a learner:

1. **One main local WordPress site for the course**
    1. Use it as your central learning sandbox.
    2. Install Bricks and create lesson pages there.
2. **Optional extra “throwaway” test sites**
    1. Use these when a lesson is highly experimental.
    2. For example, testing script loading edge cases or plugin conflicts.
3. **A dedicated child theme or code organization strategy**
    1. Even if Bricks lets you inject code in multiple places, keep your logic organized.
    2. We’ll cover that more later.
4. **PhpStorm connected to the project files**
    1. So your local files and editor reflect the actual WordPress site.
    2. This is crucial for scaling beyond tiny snippets.
5. **Browser DevTools always part of the workflow**
    1. GSAP work is visual.
    2. You need the browser open while coding, not just the editor.

Suggested folder-level mental model

Even before we get into exact project structure, you should understand the broad layers of your project.

A WordPress site typically involves:

1. **Core WordPress**
    1. The CMS itself.
    2. Usually not where you write your custom animation logic.
2. **Themes**
    1. This is often where your site presentation lives.
    2. Child themes are commonly used for safe customizations.
3. **Plugins**
    1. Third-party functionality.
    2. Sometimes custom behavior can also be placed in a plugin.

4. **Uploads / generated content**
    1. Media, cache artifacts, generated assets.
    2. Usually not where handcrafted source code should live.

For this course, your custom GSAP work will usually belong in one of these places:

1. a **child theme**,
2. a **custom code organization layer** used with Bricks,
3. in some cases a **custom plugin** for reusable site behavior.

Why local development is especially important for animation

Animation work is more sensitive than many other frontend tasks.

A regular content change either appears or does not appear.
An animation can fail in much subtler ways:

1. it runs too early,
2. it runs too late,
3. it fights CSS,
4. it stutters,
5. it causes layout shifts,
6. it works on desktop but not mobile,
7. it breaks after minification,
8. it becomes inaccessible for motion-sensitive users.

That means your environment must help you inspect:

1. **timing**,
2. **layout**,
3. **script loading**,
4. **responsive behavior**,
5. **performance**.

A local setup is where you can observe all of this safely.

Brief JavaScript insight: why environment affects JS learning □□

Since you asked for JS insights along the way, here’s an important one.

JavaScript in WordPress is not just “write code and it runs.”
Your code depends on:

1. **when it loads**,
2. **where it loads**,
3. **whether dependencies loaded first**,
4. **whether the target DOM elements exist yet**,
5. **whether another tool modifies the page after load**.

That matters a lot for GSAP.

For example, conceptually, this can fail:

```
gsap.to(".card", { y: 100 });
```

Why?

1. Maybe GSAP isn’t loaded yet.
2. Maybe `.card` doesn’t exist on that page.
3. Maybe Bricks rendered content differently than you expected.
4. Maybe your script runs before the DOM is ready.
5. Maybe CSS prevents you from seeing the movement clearly.

So your local environment is not just for “hosting WordPress.”
It is the stage on which your JavaScript execution model becomes visible.

Brief CSS insight: why environment affects CSS learning □□

GSAP often animates CSS-related properties such as:

1. `transform`,
2. `opacity`,

3. `x` / `y` equivalents through transforms,
4. scale,
5. rotation.

But CSS can interfere in subtle ways.

If an element already has layout constraints or conflicting styles, your animation may appear broken even when the JavaScript is correct.

For example:

1. a parent may have `overflow: hidden`,
2. a flex or grid layout may influence positioning,
3. an element may already be transformed,
4. transitions may conflict with GSAP,
5. responsive styles may override assumptions.

A local environment helps you inspect **computed styles**, not just your source CSS.

That’s a huge professional skill:
you stop guessing, and start verifying.

LocalWP setup strategy I recommend

You asked for detail, so here is the practical strategy I’d use if I were setting you up for this course.

Site profile

Create **one main site** specifically for this course.

Use it as your GSAP lab.

Suggested characteristics:

1. **Clean WordPress install**
    1. Avoid a bloated starter stack at first.
    2. Less noise means easier debugging.
2. **Bricks installed**
    1. This will be your page-building environment.

    2. Keep your experiments tied to actual Bricks workflows.
3. **Minimal plugin count**
    1. Install only what you need.
    2. Extra plugins increase variables and conflict potential.
4. **Readable local domain**
    1. Something like `gsap-bricks-course.local`
    2. Easy to recognize in browser tabs and tools.
5. **SSL enabled if possible**
    1. This keeps the environment closer to modern real-world browsing conditions.
    2. It also helps avoid mixed-content confusion in some cases.

Plugin philosophy for the learning environment

Keep the plugin list minimal.

A beginner mistake is installing many helper plugins too early. That creates mystery behavior.

For the course environment, use only what supports the learning goal.

Good philosophy:

1. install only what you can explain,
2. remove what you don’t actively need,
3. add tooling deliberately, not emotionally.

Why this matters:

1. animation bugs are hard enough,
2. plugin interactions multiply uncertainty,
3. clean systems teach faster.

Theme philosophy

You should aim for a setup where custom code has a clear home.

Even if Bricks allows code injection in multiple places, your long-term goal is maintainability.

At this stage, the high-level rule is:

1. use Bricks for building,

2. use a structured place for reusable custom code,
3. avoid scattering important logic randomly across pages.

We'll refine this throughout the course.

Docker setup strategy I recommend conceptually

If you go the Docker route later, your mindset should be:

1. define services clearly,
2. keep volumes understandable,
3. make project startup predictable,
4. know where your WordPress files live,
5. make sure PhpStorm edits the real mounted code,
6. keep your `.env` and compose setup readable.

In a typical conceptual Docker WordPress project, you would think in terms of:

1. **application service**
    1. Runs WordPress/PHP.
    2. Serves your site logic.
2. **database service**
    1. Stores WP content and settings.
3. **web server layer**
    1. Sometimes separate, sometimes bundled.
4. **persistent volumes**
    1. Preserve DB and/or project files between runs.
5. **port mapping**
    1. Exposes your local site to the browser.
6. **project-mounted code**
    1. Lets you edit files in PhpStorm and see them in the containerized site.

This is powerful—but it requires comfort with infrastructure vocabulary.

Comparison: LocalWP vs Docker

Here is the practical comparison for your use case.

Criteria	LocalWP	Docker
Beginner friendliness	Excellent	Moderate to difficult
Setup speed	Very fast	Slower
WordPress focus	Strong	General-purpose
Learning curve	Low	Higher
Team reproducibility	Medium	High
Environment as code	Low	High
Debugging simplicity for beginners	Better	Harder at first
Best for this course start	**Yes**	Later

So if your question is, *“What should I use now?”*
The answer is: **LocalWP**.

Your actual recommended starting stack

Here is the stack I recommend for you right now:

1. **LocalWP**
    1. For local WordPress site management.
2. **WordPress**
    1. Clean install dedicated to course exercises.
3. **BricksBuilder**
    1. Your visual building system.
4. **PhpStorm**
    1. Your code editor and project navigator.
5. **Chrome or Edge DevTools**
    1. For inspecting the DOM, CSS, network, and JS console.
6. **GSAP loaded in a controlled way**
    1. We’ll cover script loading soon.
    2. Don’t worry about the exact loading method yet.

This is enough to begin professionally without overwhelming yourself.

What “good enough” looks like at this stage

At this point, you do **not** need:

1. a perfect enterprise-grade infrastructure,
2. CI/CD pipelines,
3. a containerized multi-service architecture,
4. a headless WordPress setup,
5. advanced deployment automation.

You **do** need:

1. a stable local WordPress site,
2. Bricks working properly,
3. PhpStorm connected to your files,
4. a repeatable workflow for testing code,
5. confidence that you can experiment safely.

That is the professional beginner target.

Common beginner mistakes in local setup

1. Learning directly on a live site

This is one of the biggest mistakes.

Why it’s bad:

1. every experiment carries risk,
2. caching obscures cause and effect,
3. users may see broken states,
4. fear slows learning.

Use local first. Always.

2. Installing too many plugins

People often install optimization, helper, code snippet, animation, CSS, debug, and utility plugins all at once.

Then when something breaks, they don’t know why.

Better approach:

1. start minimal,
2. add one tool at a time,
3. verify behavior after each addition.

---

3. Mixing code locations randomly

A little code in Bricks page settings, a little in a code snippet plugin, a little in the theme, a little in the footer…

This becomes chaos quickly.

Even before we formalize architecture, keep one principle:

1. if code is reusable, give it a reusable home,
2. if code is page-specific, document where it lives,
3. if code becomes important, move it out of ad hoc locations.

---

4. Ignoring file organization because “it’s just a test”

Today’s quick test becomes tomorrow’s production pattern.

Messy local habits often become messy professional habits.

The goal is not perfection—it is **intentional structure**.

---

5. Blaming GSAP for environment problems

Sometimes GSAP is not the issue at all.

The actual problem may be:

1. scripts not loading,
2. DOM not ready,
3. selectors wrong,
4. CSS hiding the effect,
5. cache serving stale files.

A good local setup helps you identify the real cause.

Practical recommendation: your first environment plan

Here is your concrete plan for this course.

1. **Install LocalWP**
    1. Create one main course site.
    2. Give it a clear name.
2. **Install WordPress cleanly**
    1. Avoid importing a giant existing project as your first learning space.
    2. Start with clarity.
3. **Install BricksBuilder**
    1. Confirm it runs properly in the local environment.
    2. Open the builder and make sure page editing works smoothly.
4. **Open the site files in PhpStorm**
    1. We’ll discuss project structure in the next lesson.
    2. For now, the key is making the editor part of your workflow.
5. **Use the browser and DevTools from day one**
    1. Learn to keep Console and Elements tabs nearby.
    2. Animation work without DevTools is guesswork.
6. **Do not over-engineer yet**
    1. Your main job right now is to make the environment stable and understandable.
    2. Not fancy.

Mini mental model: your workflow loop

This is the core loop you will repeat many times throughout the course:

1. change code,
2. refresh page,
3. inspect behavior,
4. verify DOM/CSS/JS state,
5. adjust,
6. test again.

This loop should feel **fast**.

If your environment makes this loop slow or confusing, learning becomes harder than necessary.

Exercise: environment decision checklist

Answer these for yourself:

1. Do I want the fastest path to building GSAP projects in WP right now?
    1. If yes, choose **LocalWP**.
2. Am I currently trying to learn animation and WordPress implementation more than infrastructure engineering?
    1. If yes, choose **LocalWP**.
3. Do I need reproducible team environments and environment config in version control today?
    1. If yes, consider **Docker**.
    2. If not, Docker can wait.
4. Do I clearly understand containers, service mapping, and mounted volumes?
    1. If not, Docker may slow down early progress.

For most learners here, the result will be:
Start with LocalWP.

Implementation target for this lesson

By the end of this page, your intended setup should be:

1. **Primary environment choice**
    1. LocalWP
2. **Primary site purpose**
    1. Dedicated GSAP + Bricks learning sandbox
3. **Editor**
    1. PhpStorm
4. **Builder**
    1. BricksBuilder
5. **Testing approach**
    1. Local first, browser-driven, DevTools-assisted
6. **Advanced environment awareness**
    1. Docker understood conceptually, postponed unless needed

---

Key takeaways

1. **Your local environment is part of your skillset**, not just a technical prerequisite.
2. **LocalWP is the best starting choice** for this course because it removes setup friction.
3. **Docker is valuable**, but usually better once you are comfortable with the core WP + GSAP workflow.
4. **Animation debugging depends heavily on environment quality** because issues often involve timing, CSS, DOM state, and script loading.
5. **A clean, minimal, intentional setup will help you learn faster and more deeply**.

Action steps before moving to 1.2.2

1. Decide on your primary local setup tool:
    1. preferably **LocalWP** for now.
2. Create or plan a dedicated local WordPress site for this course.
3. Commit to this rule:

    1. *I will learn and test animations locally before touching live sites.*
4. Make sure you have:
    1. LocalWP,
    2. PhpStorm,
    3. a modern browser with DevTools,
    4. access to BricksBuilder.

Revision #1
Created 2026-04-21 17:17:12 UTC by art10m
Updated 2026-04-21 17:22:25 UTC by art10m