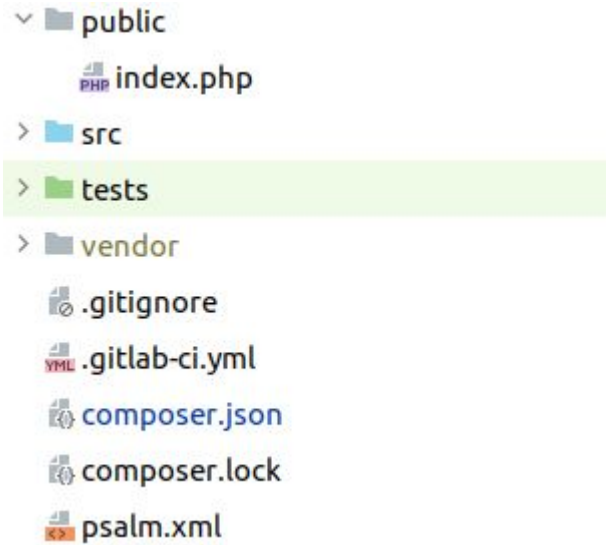
Projektaufbau

Ordnerstruktur



.gitignore

- **Git vergisst nicht! (oder nur sehr wenig)**
- Einige Dinge gehöre nicht in die git Historie
  - Lassen sich über *.gitignore* Dateien definieren
- Nicht in die Historie gehören z.B.
  - IDE Meta Daten (Z.B. *.idea* oder *.vscode* Ordner)
  - Lokale Konfigurationsdateien wie *.env* (**Achtung: Sicherheitsrisiko**)
  - Pakete und Abhängigkeiten (*vendor*, *node_modules*, *libs*, etc.)
  - Inhalte von Caches
  - Gebaute Artefakte
  - ...

```
# IDE meta data
.idea
.vscode

# Dependencies
vendor
node_modules

# Local caches
cache/*
!cache/.gitkeep
```

Dependency Management

- Package Manager für PHP
  - Open Source
  - Quasi Monopol
- Lädt Pakete aus verschiedenen Quellen
- Nutzt **Semantic Versioning**
  - Stark verbesserte Inter-Paket Operabilität
- Übernimmt Autoloading von Klassen
- JSON Konfiguration

Dependency Management

- Composer sollte Teil jedes PHP Projektes sein
  - Zugriff auf tausende von guten Bibliotheken
  - Standardisiert Metainformation über eigenes Projekt
- Konfiguration über *composer.json* Datei
  - Locking der aktuell installierten Abhängigkeiten über *composer.lock* Datei
- Bedienbar über CLI Interface
- Ähnliche Tools für quasi alle anderen Sprachen vorhanden
  - Empfehlung für Javascript: *yarn*

Autoloading

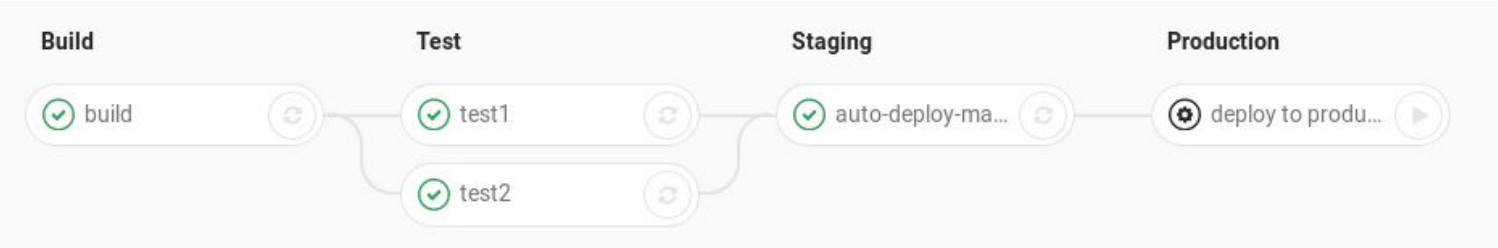
- Woher kennt Laufzeitumgebung die Klassendefinitionen?
- Technisch getrennte Welten
  - Filesystem (Dateien mit Code)
  - Namespace Hierarchie
- Autoloading in PHP im Userland
  - Eigentlich selbst zu leisten
  - Composer bietet technische Lösung
- Bitte PSR-4 Standard nutzen!
  - Verbindet Ordnerstruktur 1:1 mit Namespaces
  - Bietet “Einstiegspunkt” in Namespaces

composer.json

```
{
  "name": "wickb/webentwicklung",
  "type": "project",
  "description": "Project stub für das FWPM 'Webentwicklung'",
  "authors": [
    {
      "name": "Bernhard Wick",
      "email": "bernhard.wick@th-rosenheim.de"
    }
  ],
  "require": {
     "php": ">=7.3"
  },
  "require-dev": {
    "phpunit/phpunit": "^9", 9.5.3
    "squizlabs/php_codesniffer": "3.*", 3.5.8
    "vimeo/psalm": "^3.11" 3.18.2
  },
  "autoload": {
    "psr-4": {
      "Wickb\\Webentwicklung\\": "src/"
    }
  }
}
```

Gitlab

- Git basierte Code Hosting Plattform mit integrierten CI Kapazitäten
  - Auch als SAAS Lösung nutzbar
  - Voll integriert dank doppelter Funktion
- Leicht erweiterbar durch “Runner” und “Executor” Konzept
  - Kann fast beliebiges System nachstellen
- Teilt Pipeline in “Jobs” und “Stages”
  - Erlaubt bedingte Ausführung und Parallelisierung
- Erlaubt unterschiedliche Pipelines pro Projekt



Gitlab - .gitlab-ci.yml

- Konfigurationsdatei der Gitlab Pipeline
  - “Infrastructure as Code”
- YAML Format
- Sehr feingranular konfigurierbar
  - Einrückung/Nesting ergibt Kontext
- Unterteilung innerhalb von Jobs möglich
- Kann Job-Ergebnisse zwischenspeichern
  - Sog. Artefakte

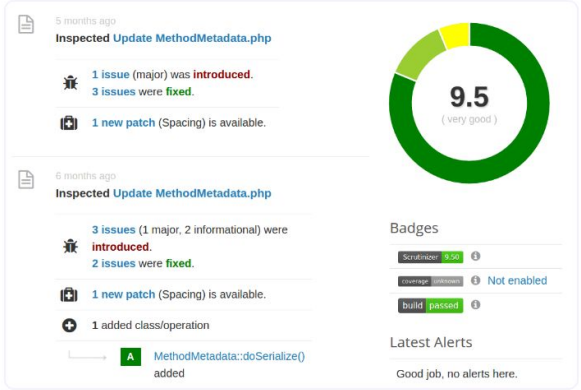
```yaml
stages:
  - build
  - deploy

image: php:7.3

before_script:
  - echo "Going to the next stage"

build:
  stage: build
  script:
    - "php -v"

deploy:
  stage: deploy
  script:
    - "echo Deploying!"
```

Statische Code Analysen/Linting

- Ursprünglich als Compiler Unterstützung
  - Erstes Tool namens *Lint* 1970er Jahre für C
- Versuchen “Code Smells” und unsaubere Architekturmuster aufzudecken
  - Liefern Hinweise zur Verbesserung von Code
- Helfen bei Optimierung und Vermeiden technischer Schuld
- Sichern Qualität von Code ab
  - Decken ein weites Feld an Metriken ab
- Schützen vor banalen Fehlern, z.B. Syntaxfehler

Statische Code Analysen/Linting

- Beispiele:
  - php -l - Testet auf Syntaxfehler
  - phpcpd - Deckt Redundanz auf
  - Psalm - Prüft auf Sammlung von Code Smells
  - dephpend - Architekturanalyse und Abhängigkeitsvisualisierung
  - Liste unter https://github.com/exakat/php-static-analysis-tools
  - ESLint - Sammlung an Javascript Metriken
- Viele wirklich gute SAAS Tools
  - Scrutinizer
  - Blackfire.io
  - SymfonyInsight



Log Files und Konsolen

- Geben Aufschluss über aufgetretene Fehler
- Überwachung meistens nur manuell und nach Wahrnehmung eines Fehlerfalls
  - Automatisches Monitoring und Reaktion bei erstem Auftreten des Fehlerfalls wünschenswert
- Abhängig vom Ort an dem Fehler auftritt!
  - Im Browser, im Webserver, im Framework (gefangene Exceptions), in der PHP Runtime, …
    - Jeweils unterschiedliche Logging Systeme!
- Relevanteste Log Files und Konsolen
  - PHP Laufzeitumgebung: */tmp/php_errors.log* (oder je nach Konfiguration)
  - JS Laufzeitumgebung: Entwicklerkonsole im Browser

Quellen:

- https://git-scm.com/docs/githooks
- https://google.github.io/styleguide/jsguide.html
- https://standardjs.com/
- https://google.github.io/styleguide/htmlcssguide.html
- https://stateofjs.com/
- https://github.com/exakat/php-static-analysis-tools
- https://docs.gitlab.com/ee/ci/examples/php.html

Bildquellen:

- Wraith Beispiel https://pantheon.io/docs/guides/visual-diff-with-wraith
- Pipelines Beispiel https://docs.gitlab.com/ee/ci/pipelines/index.html