Folder api

27 printable files

(file list disabled)

api/composer.json

```
{
    "name": "fhtechnikum/uebung3_4",
    "autoload": {
        "psr-4": {
            "Fhtechnikum\\Uebung34\\": "src/"
        }
    },
    "authors": [
        {
            "name": "fhtechnikum",
            "email": "lammer@technikum-wien.at"
        }
    ],
    "require": {}
}
```

api/composer.lock

```
{
    "_readme": [
        "This file locks the dependencies of your project to a known state",
        "Read more about it at https://getcomposer.org/doc/01-basic-
usage.md#installing-dependencies",
        "This file is @generated automatically"
    ],
    "content-hash": "fb6e6a93cf8f56dc83a4706e50bd0562",
    "packages": [],
    "packages-dev": [],
    "aliases": [],
    "minimum-stability": "stable",
    "stability-flags": [],
    "prefer-stable": false,
    "prefer-lowest": false,
    "platform": [],
    "platform-dev": [],
    "plugin-api-version": "2.3.0"
}
```

api/index.php

```
<?php
```

```php
use Fhtechnikum\Uebung34\Controller\ProductsListController;

error_reporting(E_ERROR);
ini_set("display_errors", 1);
include 'src/config/config.php';
require 'vendor/autoload.php';

$controller = new ProductsListController();
$controller->route();
```

api/src/Controller/ProductsListController.php

```php
<?php
namespace Fhtechnikum\Uebung34\Controller;
use Fhtechnikum\Uebung34\DTOs\ProductDTO;
use Fhtechnikum\Uebung34\DTOs\ProductListDTO;
use Fhtechnikum\Uebung34\DTOs\ProductTypeDTO;
use Fhtechnikum\Uebung34\Gateways\ProductsReadDBGateway;
use Fhtechnikum\Uebung34\Services\ProductsService;
use Fhtechnikum\Uebung34\Views\JsonView;

class ProductsListController
{

    private $service;
    private $jsonView;
    private $url = API_URL;

    public function __construct()
    {
        $gateway = new ProductsReadDBGateway(
            DBHost,
            DBName,
            DBUsername,
            DBPassword
        );
        $this->service = new ProductsService($gateway);
        $this->jsonView = new JsonView();
    }

    public function route()
    {
        $action = filter_input(INPUT_GET, "action", FILTER_SANITIZE_STRING);
        switch(strtolower($action)) {
            case 'listtypes': //list product types
                $this->listProductTypes();
                break;
            case 'listproductsbytypeid': //list products by type id
                $productTypeId = filter_input(INPUT_GET, "typeId",
FILTER_SANITIZE_NUMBER_INT);
                if(!$productTypeId) {
                    return $this->error("Invalid Type Id");
                }
                $this->listProductsByTypeId($productTypeId);
                break;
            default:
```

```
                $this->error("Unknown Action");
            }
        }

        private function listProductTypes()
        {
            $productTypesList = $this->service->getAllProductTypes();
            $dtoList = [];
            foreach($productTypesList as $item) {
                $dtoList[] = ProductTypeDTO::map($item, $this->url);
            }

            $this->jsonView->display($dtoList);
        }

        private function listProductsByTypeId($productTypeId)
        {
            $productsList = $this->service->getProductsByTypeId($productTypeId);
            $productTypeName = $this->service->getProductTypeName($productTypeId);
            $dtoList = [];
            foreach($productsList as $item) {
                $dtoList[] = ProductDTO::map($item);
            }

            $response = new ProductListDTO();
            $response->productType = $productTypeName;
            $response->products = $dtoList;
            $response->url = $this->url . "?action=listTypes";

            $this->jsonView->display($response);
        }

        private function error($errorMessage)
        {
            //display via view ... the error
            print ($errorMessage);
        }
    }
```

api/src/DTOs/ProductDTO.php

```
    <?php

    namespace Fhtechnikum\Uebung34\DTOs;

    class ProductDTO
    {
        public $name;
        public $id;

        public static function map($productModel) {

            $dto = new ProductDTO();
            $dto->name = $productModel->name;
            $dto->id = $productModel->id;
```

```
        return $dto;
    }
}
```

api/src/DTOs/ProductListDTO.php

```
<?php

namespace Fhtechnikum\Uebung34\DTOs;

class ProductListDTO
{
    public $productType;
    public $products;
    public $url;
}
```

api/src/DTOs/ProductTypeDTO.php

```
<?php

namespace Fhtechnikum\Uebung34\DTOs;

class ProductTypeDTO
{
    public $productType;
    public $url;

    public static function map($productTypeModel, $url) {

        $productTypeDTO = new ProductTypeDTO();
        $productTypeDTO->productType = $productTypeModel->name;
        $productTypeDTO->url = $url . "?action=listProductsByTypeId&typeId=" .
$productTypeModel->id;

        return $productTypeDTO;
    }
}
```

api/src/Gateways/ProductsReadDBGateway.php

```
<?php

namespace Fhtechnikum\Uebung34\Gateways;

use Fhtechnikum\Uebung34\Models\ProductModel;
use Fhtechnikum\Uebung34\Models\ProductTypeModel;
use PDO;

class ProductsReadDBGateway implements ProductsReadGatewayInterface
{
    private $pdo;
```

```php
    public function __construct(
        $host,
        $dbname,
        $user,
        $password
    )
    {
        $this->pdo = new PDO('mysql:host='.$host.';dbname='.$dbname, $user,
$password);
    }

    public function getAllProductTypes()
    {
        $sql = "SELECT id, name FROM product_types ORDER BY name";
        $statement = $this->pdo->prepare($sql);
        $statement->execute();

        $productTypesList = $statement->fetchAll(PDO::FETCH_CLASS);

        return $this->mapProductTypes($productTypesList);
    }

    private function mapProductTypes(array $productTypesList)
    {
        $return = [];
        foreach ($productTypesList as $productType) {

            $productTypeModel = new ProductTypeModel();
            $productTypeModel->name = $productType->name;
            $productTypeModel->id = $productType->id;
            $return[] = $productTypeModel;
        }

        return $return;
    }

    public function getProductsByTypeId($productTypeId) {
        $sql = "SELECT name, id FROM products
            WHERE id_product_types = :productTypeId";
        $statement = $this->pdo->prepare($sql);
        $statement->bindParam(':productTypeId', $productTypeId);
        $statement->execute();

        $productList = $statement->fetchAll(PDO::FETCH_CLASS);

        return $this->mapProduct($productList);
    }

    private function mapProduct(array $productList)
    {
        $return = [];
        foreach ($productList as $product) {
            $productModel = new ProductModel();
            $productModel->name = $product->name;
            $productModel->id = $product->id;
            $return[] = $productModel;
        }

        return $return;
    }
```

```
    public function getProductTypeName($productTypeId) {
        $sql = "SELECT name FROM product_types
            WHERE id = :productTypeId";
        $statement = $this->pdo->prepare($sql);
        $statement->bindParam(':productTypeId', $productTypeId);
        $statement->execute();

        $productType = $statement->fetch(PDO::FETCH_OBJ);
        //valid result?
        return $productType->name;
    }

}
```

api/src/Gateways/ProductsReadGatewayInterface.php

```
<?php

namespace Fhtechnikum\Uebung34\Gateways;

interface ProductsReadGatewayInterface
{
    public function getAllProductTypes();
}
```

api/src/Models/ProductModel.php

```
<?php

namespace Fhtechnikum\Uebung34\Models;

class ProductModel
{
    public $name;
    public $id;
}
```

api/src/Models/ProductTypeModel.php

```
<?php
namespace Fhtechnikum\Uebung34\Models;
class ProductTypeModel
{
    public $name;
    public $id;
}
```

api/src/Services/ProductsService.php

```php
<?php

namespace Fhtechnikum\Uebung34\Services;

class ProductsService
{
    private $productsReadGateway;

    public function __construct($productsReadGateway)
    {
        $this->productsReadGateway = $productsReadGateway;
    }


    public function getAllProductTypes()
    {
        $productModelList = $this->productsReadGateway->getAllProductTypes();
        return $productModelList;
    }

    public function getProductsByTypeId($productTypeId)
    {
        $productList = $this->productsReadGateway-
>getProductsByTypeId($productTypeId);
        return $productList;
    }

    public function getProductTypeName($productTypeId)
    {
        $productTypeName = $this->productsReadGateway-
>getProductTypeName($productTypeId);
        return $productTypeName;
    }
}
```

api/src/Views/HTMLTableView.php

```php
<?php

namespace Fhtechnikum\Uebung34\Views;

class HTMLTableView implements ViewInterface
{

    public function display($data)
    {
        echo "<table border='1'>";
        $first = true;
        foreach($data as $name => $entry) {
            if($first) {
                echo "<thead><tr>";
                echo "<th>" . $name . "</th>";
                echo "</tr></thead>";
                $first = false;
            }

            if($name == "products"){
                echo "<td>";
```

```
                $this->displayProductsList($entry);
                echo "</td>";
            } else {
                echo "<td>"  . $entry . "</td>";
            }
        }
        echo "</table>";

    }

    private function displayProductsList($productList)
    {
        echo "<ul>";

        foreach($productList as $item) {

            echo "<li>" . $item->name ."</li>";
        }

        echo "</ul>";
    }
}
```

api/src/Views/JsonView.php

```
<?php

namespace Fhtechnikum\Uebung34\Views;

class JsonView implements ViewInterface
{

    public function display($data)
    {
        header("Content-Type: application/json");
        echo json_encode($data);
    }
}
```

api/src/Views/ViewInterface.php

```
<?php

namespace Fhtechnikum\Uebung34\Views;

interface ViewInterface
{
    public function display($data);
}
```

api/src/config/config.php

```php
<?php
define("DBHost", "localhost");
define("DBName", "fh_products_2023");
define("DBPassword", "z4omLWuIHKnbacCo");
define("DBUsername", "fh");

define("API_URL", "http://students.fh/academy/backendBasics/ss2023/Uebung3_4/api/");
```

api/vendor/autoload.php

```php
<?php

// autoload.php @generated by Composer

if (PHP_VERSION_ID < 50600) {
    if (!headers_sent()) {
        header('HTTP/1.1 500 Internal Server Error');
    }
    $err = 'Composer 2.3.0 dropped support for autoloading on PHP <5.6 and you are
running '.PHP_VERSION.', please upgrade PHP or use Composer 2.2 LTS via "composer
self-update --2.2". Aborting.'.PHP_EOL;
    if (!ini_get('display_errors')) {
        if (PHP_SAPI === 'cli' || PHP_SAPI === 'phpdbg') {
            fwrite(STDERR, $err);
        } elseif (!headers_sent()) {
            echo $err;
        }
    }
    trigger_error(
        $err,
        E_USER_ERROR
    );
}

require_once __DIR__ . '/composer/autoload_real.php';

return ComposerAutoloaderInit1fd71056277afaa2ee52250b805bde15::getLoader();
```

api/vendor/composer/ClassLoader.php

```php
<?php

/*
 * This file is part of Composer.
 *
 * (c) Nils Adermann <naderman@naderman.de>
 *     Jordi Boggiano <j.boggiano@seld.be>
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Composer\Autoload;
```

```php
/**
 * ClassLoader implements a PSR-0, PSR-4 and classmap class loader.
 *
 *     $loader = new \Composer\Autoload\ClassLoader();
 *
 *     // register classes with namespaces
 *     $loader->add('Symfony\Component', __DIR__.'/component');
 *     $loader->add('Symfony',           __DIR__.'/framework');
 *
 *     // activate the autoloader
 *     $loader->register();
 *
 *     // to enable searching the include path (eg. for PEAR packages)
 *     $loader->setUseIncludePath(true);
 *
 * In this example, if you try to use a class in the Symfony\Component
 * namespace or one of its children (Symfony\Component\Console for instance),
 * the autoloader will first look for the class under the component/
 * directory, and it will then fallback to the framework/ directory if not
 * found before giving up.
 *
 * This class is loosely based on the Symfony UniversalClassLoader.
 *
 * @author Fabien Potencier <fabien@symfony.com>
 * @author Jordi Boggiano <j.boggiano@seld.be>
 * @see    https://www.php-fig.org/psr/psr-0/
 * @see    https://www.php-fig.org/psr/psr-4/
 */
class ClassLoader
{
    /** @var ?string */
    private $vendorDir;

    // PSR-4
    /**
     * @var array[]
     * @psalm-var array<string, array<string, int>>
     */
    private $prefixLengthsPsr4 = array();
    /**
     * @var array[]
     * @psalm-var array<string, array<int, string>>
     */
    private $prefixDirsPsr4 = array();
    /**
     * @var array[]
     * @psalm-var array<string, string>
     */
    private $fallbackDirsPsr4 = array();

    // PSR-0
    /**
     * @var array[]
     * @psalm-var array<string, array<string, string[]>>
     */
    private $prefixesPsr0 = array();
    /**
     * @var array[]
     * @psalm-var array<string, string>
     */
```

```
    private $fallbackDirsPsr0 = array();

    /** @var bool */
    private $useIncludePath = false;

    /**
     * @var string[]
     * @psalm-var array<string, string>
     */
    private $classMap = array();

    /** @var bool */
    private $classMapAuthoritative = false;

    /**
     * @var bool[]
     * @psalm-var array<string, bool>
     */
    private $missingClasses = array();

    /** @var ?string */
    private $apcuPrefix;

    /**
     * @var self[]
     */
    private static $registeredLoaders = array();

    /**
     * @param ?string $vendorDir
     */
    public function __construct($vendorDir = null)
    {
        $this->vendorDir = $vendorDir;
    }

    /**
     * @return string[]
     */
    public function getPrefixes()
    {
        if (!empty($this->prefixesPsr0)) {
            return call_user_func_array('array_merge', array_values($this->prefixesPsr0));
        }

        return array();
    }

    /**
     * @return array[]
     * @psalm-return array<string, array<int, string>>
     */
    public function getPrefixesPsr4()
    {
        return $this->prefixDirsPsr4;
    }

    /**
     * @return array[]
```

```
     * @psalm-return array<string, string>
     */
    public function getFallbackDirs()
    {
        return $this->fallbackDirsPsr0;
    }

    /**
     * @return array[]
     * @psalm-return array<string, string>
     */
    public function getFallbackDirsPsr4()
    {
        return $this->fallbackDirsPsr4;
    }

    /**
     * @return string[] Array of classname => path
     * @psalm-return array<string, string>
     */
    public function getClassMap()
    {
        return $this->classMap;
    }

    /**
     * @param string[] $classMap Class to filename map
     * @psalm-param array<string, string> $classMap
     *
     * @return void
     */
    public function addClassMap(array $classMap)
    {
        if ($this->classMap) {
            $this->classMap = array_merge($this->classMap, $classMap);
        } else {
            $this->classMap = $classMap;
        }
    }

    /**
     * Registers a set of PSR-0 directories for a given prefix, either
     * appending or prepending to the ones previously set for this prefix.
     *
     * @param string          $prefix  The prefix
     * @param string[]|string $paths   The PSR-0 root directories
     * @param bool            $prepend Whether to prepend the directories
     *
     * @return void
     */
    public function add($prefix, $paths, $prepend = false)
    {
        if (!$prefix) {
            if ($prepend) {
                $this->fallbackDirsPsr0 = array_merge(
                    (array) $paths,
                    $this->fallbackDirsPsr0
                );
            } else {
                $this->fallbackDirsPsr0 = array_merge(
```

```
                    $this->fallbackDirsPsr0,
                    (array) $paths
                );
            }

            return;
        }

        $first = $prefix[0];
        if (!isset($this->prefixesPsr0[$first][$prefix])) {
            $this->prefixesPsr0[$first][$prefix] = (array) $paths;

            return;
        }
        if ($prepend) {
            $this->prefixesPsr0[$first][$prefix] = array_merge(
                (array) $paths,
                $this->prefixesPsr0[$first][$prefix]
            );
        } else {
            $this->prefixesPsr0[$first][$prefix] = array_merge(
                $this->prefixesPsr0[$first][$prefix],
                (array) $paths
            );
        }
    }

    /**
     * Registers a set of PSR-4 directories for a given namespace, either
     * appending or prepending to the ones previously set for this namespace.
     *
     * @param string          $prefix  The prefix/namespace, with trailing '\\'
     * @param string[]|string $paths   The PSR-4 base directories
     * @param bool            $prepend Whether to prepend the directories
     *
     * @throws \InvalidArgumentException
     *
     * @return void
     */
    public function addPsr4($prefix, $paths, $prepend = false)
    {
        if (!$prefix) {
            // Register directories for the root namespace.
            if ($prepend) {
                $this->fallbackDirsPsr4 = array_merge(
                    (array) $paths,
                    $this->fallbackDirsPsr4
                );
            } else {
                $this->fallbackDirsPsr4 = array_merge(
                    $this->fallbackDirsPsr4,
                    (array) $paths
                );
            }
        } elseif (!isset($this->prefixDirsPsr4[$prefix])) {
            // Register directories for a new namespace.
            $length = strlen($prefix);
            if ('\\' !== $prefix[$length - 1]) {
                throw new \InvalidArgumentException("A non-empty PSR-4 prefix must end
with a namespace separator.");
```

```
            }
            $this->prefixLengthsPsr4[$prefix[0]][$prefix] = $length;
            $this->prefixDirsPsr4[$prefix] = (array) $paths;
        } elseif ($prepend) {
            // Prepend directories for an already registered namespace.
            $this->prefixDirsPsr4[$prefix] = array_merge(
                (array) $paths,
                $this->prefixDirsPsr4[$prefix]
            );
        } else {
            // Append directories for an already registered namespace.
            $this->prefixDirsPsr4[$prefix] = array_merge(
                $this->prefixDirsPsr4[$prefix],
                (array) $paths
            );
        }
    }

    /**
     * Registers a set of PSR-0 directories for a given prefix,
     * replacing any others previously set for this prefix.
     *
     * @param string          $prefix The prefix
     * @param string[]|string $paths  The PSR-0 base directories
     *
     * @return void
     */
    public function set($prefix, $paths)
    {
        if (!$prefix) {
            $this->fallbackDirsPsr0 = (array) $paths;
        } else {
            $this->prefixesPsr0[$prefix[0]][$prefix] = (array) $paths;
        }
    }

    /**
     * Registers a set of PSR-4 directories for a given namespace,
     * replacing any others previously set for this namespace.
     *
     * @param string          $prefix The prefix/namespace, with trailing '\\'
     * @param string[]|string $paths  The PSR-4 base directories
     *
     * @throws \InvalidArgumentException
     *
     * @return void
     */
    public function setPsr4($prefix, $paths)
    {
        if (!$prefix) {
            $this->fallbackDirsPsr4 = (array) $paths;
        } else {
            $length = strlen($prefix);
            if ('\\' !== $prefix[$length - 1]) {
                throw new \InvalidArgumentException("A non-empty PSR-4 prefix must end
with a namespace separator.");
            }
            $this->prefixLengthsPsr4[$prefix[0]][$prefix] = $length;
            $this->prefixDirsPsr4[$prefix] = (array) $paths;
        }
```

```
    }

    /**
     * Turns on searching the include path for class files.
     *
     * @param bool $useIncludePath
     *
     * @return void
     */
    public function setUseIncludePath($useIncludePath)
    {
        $this->useIncludePath = $useIncludePath;
    }

    /**
     * Can be used to check if the autoloader uses the include path to check
     * for classes.
     *
     * @return bool
     */
    public function getUseIncludePath()
    {
        return $this->useIncludePath;
    }

    /**
     * Turns off searching the prefix and fallback directories for classes
     * that have not been registered with the class map.
     *
     * @param bool $classMapAuthoritative
     *
     * @return void
     */
    public function setClassMapAuthoritative($classMapAuthoritative)
    {
        $this->classMapAuthoritative = $classMapAuthoritative;
    }

    /**
     * Should class lookup fail if not found in the current class map?
     *
     * @return bool
     */
    public function isClassMapAuthoritative()
    {
        return $this->classMapAuthoritative;
    }

    /**
     * APCu prefix to use to cache found/not-found classes, if the extension is
enabled.
     *
     * @param string|null $apcuPrefix
     *
     * @return void
     */
    public function setApcuPrefix($apcuPrefix)
    {
        $this->apcuPrefix = function_exists('apcu_fetch') &&
filter_var(ini_get('apc.enabled'), FILTER_VALIDATE_BOOLEAN) ? $apcuPrefix : null;
```

```
    }

    /**
     * The APCu prefix in use, or null if APCu caching is not enabled.
     *
     * @return string|null
     */
    public function getApcuPrefix()
    {
        return $this->apcuPrefix;
    }

    /**
     * Registers this instance as an autoloader.
     *
     * @param bool $prepend Whether to prepend the autoloader or not
     *
     * @return void
     */
    public function register($prepend = false)
    {
        spl_autoload_register(array($this, 'loadClass'), true, $prepend);

        if (null === $this->vendorDir) {
            return;
        }

        if ($prepend) {
            self::$registeredLoaders = array($this->vendorDir => $this) +
self::$registeredLoaders;
        } else {
            unset(self::$registeredLoaders[$this->vendorDir]);
            self::$registeredLoaders[$this->vendorDir] = $this;
        }
    }

    /**
     * Unregisters this instance as an autoloader.
     *
     * @return void
     */
    public function unregister()
    {
        spl_autoload_unregister(array($this, 'loadClass'));

        if (null !== $this->vendorDir) {
            unset(self::$registeredLoaders[$this->vendorDir]);
        }
    }

    /**
     * Loads the given class or interface.
     *
     * @param  string    $class The name of the class
     * @return true|null True if loaded, null otherwise
     */
    public function loadClass($class)
    {
        if ($file = $this->findFile($class)) {
            includeFile($file);
```

```
            return true;
        }

        return null;
    }

    /**
     * Finds the path to the file where the class is defined.
     *
     * @param string $class The name of the class
     *
     * @return string|false The path if found, false otherwise
     */
    public function findFile($class)
    {
        // class map lookup
        if (isset($this->classMap[$class])) {
            return $this->classMap[$class];
        }
        if ($this->classMapAuthoritative || isset($this->missingClasses[$class])) {
            return false;
        }
        if (null !== $this->apcuPrefix) {
            $file = apcu_fetch($this->apcuPrefix.$class, $hit);
            if ($hit) {
                return $file;
            }
        }

        $file = $this->findFileWithExtension($class, '.php');

        // Search for Hack files if we are running on HHVM
        if (false === $file && defined('HHVM_VERSION')) {
            $file = $this->findFileWithExtension($class, '.hh');
        }

        if (null !== $this->apcuPrefix) {
            apcu_add($this->apcuPrefix.$class, $file);
        }

        if (false === $file) {
            // Remember that this class does not exist.
            $this->missingClasses[$class] = true;
        }

        return $file;
    }

    /**
     * Returns the currently registered loaders indexed by their corresponding vendor
directories.
     *
     * @return self[]
     */
    public static function getRegisteredLoaders()
    {
        return self::$registeredLoaders;
    }
```

```php
    /**
     * @param  string       $class
     * @param  string       $ext
     * @return string|false
     */
    private function findFileWithExtension($class, $ext)
    {
        // PSR-4 lookup
        $logicalPathPsr4 = strtr($class, '\\', DIRECTORY_SEPARATOR) . $ext;

        $first = $class[0];
        if (isset($this->prefixLengthsPsr4[$first])) {
            $subPath = $class;
            while (false !== $lastPos = strrpos($subPath, '\\')) {
                $subPath = substr($subPath, 0, $lastPos);
                $search = $subPath . '\\';
                if (isset($this->prefixDirsPsr4[$search])) {
                    $pathEnd = DIRECTORY_SEPARATOR . substr($logicalPathPsr4, $lastPos
+ 1);
                    foreach ($this->prefixDirsPsr4[$search] as $dir) {
                        if (file_exists($file = $dir . $pathEnd)) {
                            return $file;
                        }
                    }
                }
            }
        }

        // PSR-4 fallback dirs
        foreach ($this->fallbackDirsPsr4 as $dir) {
            if (file_exists($file = $dir . DIRECTORY_SEPARATOR . $logicalPathPsr4)) {
                return $file;
            }
        }

        // PSR-0 lookup
        if (false !== $pos = strrpos($class, '\\')) {
            // namespaced class name
            $logicalPathPsr0 = substr($logicalPathPsr4, 0, $pos + 1)
                . strtr(substr($logicalPathPsr4, $pos + 1), '_', DIRECTORY_SEPARATOR);
        } else {
            // PEAR-like class name
            $logicalPathPsr0 = strtr($class, '_', DIRECTORY_SEPARATOR) . $ext;
        }

        if (isset($this->prefixesPsr0[$first])) {
            foreach ($this->prefixesPsr0[$first] as $prefix => $dirs) {
                if (0 === strpos($class, $prefix)) {
                    foreach ($dirs as $dir) {
                        if (file_exists($file = $dir . DIRECTORY_SEPARATOR .
$logicalPathPsr0)) {
                            return $file;
                        }
                    }
                }
            }
        }

        // PSR-0 fallback dirs
        foreach ($this->fallbackDirsPsr0 as $dir) {
```

```
            if (file_exists($file = $dir . DIRECTORY_SEPARATOR . $logicalPathPsr0)) {
                return $file;
            }
        }

        // PSR-0 include paths.
        if ($this->useIncludePath && $file =
stream_resolve_include_path($logicalPathPsr0)) {
            return $file;
        }

        return false;
    }
}

/**
 * Scope isolated include.
 *
 * Prevents access to $this/self from included files.
 *
 * @param  string $file
 * @return void
 * @private
 */
function includeFile($file)
{
    include $file;
}
```

api/vendor/composer/InstalledVersions.php

```
<?php

/*
 * This file is part of Composer.
 *
 * (c) Nils Adermann <naderman@naderman.de>
 *     Jordi Boggiano <j.boggiano@seld.be>
 *
 * For the full copyright and license information, please view the LICENSE
 * file that was distributed with this source code.
 */

namespace Composer;

use Composer\Autoload\ClassLoader;
use Composer\Semver\VersionParser;

/**
 * This class is copied in every Composer installed project and available to all
 *
 * See also https://getcomposer.org/doc/07-runtime.md#installed-versions
 *
 * To require its presence, you can require `composer-runtime-api ^2.0`
 *
```

```
 * @final
 */
class InstalledVersions
{
    /**
     * @var mixed[]|null
     * @psalm-var array{root: array{name: string, pretty_version: string, version:
string, reference: string|null, type: string, install_path: string, aliases: string[],
dev: bool}, versions: array<string, array{pretty_version?: string, version?: string,
reference?: string|null, type?: string, install_path?: string, aliases?: string[],
dev_requirement: bool, replaced?: string[], provided?: string[]}>}|array{}|null
     */
    private static $installed;

    /**
     * @var bool|null
     */
    private static $canGetVendors;

    /**
     * @var array[]
     * @psalm-var array<string, array{root: array{name: string, pretty_version:
string, version: string, reference: string|null, type: string, install_path: string,
aliases: string[], dev: bool}, versions: array<string, array{pretty_version?: string,
version?: string, reference?: string|null, type?: string, install_path?: string,
aliases?: string[], dev_requirement: bool, replaced?: string[], provided?:
string[]}>}>
     */
    private static $installedByVendor = array();

    /**
     * Returns a list of all package names which are present, either by being
installed, replaced or provided
     *
     * @return string[]
     * @psalm-return list<string>
     */
    public static function getInstalledPackages()
    {
        $packages = array();
        foreach (self::getInstalled() as $installed) {
            $packages[] = array_keys($installed['versions']);
        }

        if (1 === \count($packages)) {
            return $packages[0];
        }

        return array_keys(array_flip(\call_user_func_array('array_merge',
$packages)));
    }

    /**
     * Returns a list of all package names with a specific type e.g. 'library'
     *
     * @param  string   $type
     * @return string[]
     * @psalm-return list<string>
     */
    public static function getInstalledPackagesByType($type)
    {
        $packagesByType = array();
```

```
        foreach (self::getInstalled() as $installed) {
            foreach ($installed['versions'] as $name => $package) {
                if (isset($package['type']) && $package['type'] === $type) {
                    $packagesByType[] = $name;
                }
            }
        }

        return $packagesByType;
    }

    /**
     * Checks whether the given package is installed
     *
     * This also returns true if the package name is provided or replaced by another
package
     *
     * @param  string $packageName
     * @param  bool   $includeDevRequirements
     * @return bool
     */
    public static function isInstalled($packageName, $includeDevRequirements = true)
    {
        foreach (self::getInstalled() as $installed) {
            if (isset($installed['versions'][$packageName])) {
                return $includeDevRequirements || empty($installed['versions']
[$packageName]['dev_requirement']);
            }
        }

        return false;
    }

    /**
     * Checks whether the given package satisfies a version constraint
     *
     * e.g. If you want to know whether version 2.3+ of package foo/bar is installed,
you would call:
     *
     *   Composer\InstalledVersions::satisfies(new VersionParser, 'foo/bar', '^2.3')
     *
     * @param  VersionParser $parser      Install composer/semver to have access to
this class and functionality
     * @param  string        $packageName
     * @param  string|null   $constraint  A version constraint to check for, if you
pass one you have to make sure composer/semver is required by your package
     * @return bool
     */
    public static function satisfies(VersionParser $parser, $packageName, $constraint)
    {
        $constraint = $parser->parseConstraints($constraint);
        $provided = $parser->parseConstraints(self::getVersionRanges($packageName));

        return $provided->matches($constraint);
    }

    /**
     * Returns a version constraint representing all the range(s) which are installed
for a given package
     *
     * It is easier to use this via isInstalled() with the $constraint argument if you
need to check
```

```
     * whether a given version of a package is installed, and not just whether it
exists
     *
     * @param  string $packageName
     * @return string Version constraint usable with composer/semver
     */
    public static function getVersionRanges($packageName)
    {
        foreach (self::getInstalled() as $installed) {
            if (!isset($installed['versions'][$packageName])) {
                continue;
            }

            $ranges = array();
            if (isset($installed['versions'][$packageName]['pretty_version'])) {
                $ranges[] = $installed['versions'][$packageName]['pretty_version'];
            }
            if (array_key_exists('aliases', $installed['versions'][$packageName])) {
                $ranges = array_merge($ranges, $installed['versions'][$packageName]
['aliases']);
            }
            if (array_key_exists('replaced', $installed['versions'][$packageName])) {
                $ranges = array_merge($ranges, $installed['versions'][$packageName]
['replaced']);
            }
            if (array_key_exists('provided', $installed['versions'][$packageName])) {
                $ranges = array_merge($ranges, $installed['versions'][$packageName]
['provided']);
            }

            return implode(' || ', $ranges);
        }

        throw new \OutOfBoundsException('Package "' . $packageName . '" is not
installed');
    }

    /**
     * @param  string      $packageName
     * @return string|null If the package is being replaced or provided but is not
really installed, null will be returned as version, use satisfies or getVersionRanges
if you need to know if a given version is present
     */
    public static function getVersion($packageName)
    {
        foreach (self::getInstalled() as $installed) {
            if (!isset($installed['versions'][$packageName])) {
                continue;
            }

            if (!isset($installed['versions'][$packageName]['version'])) {
                return null;
            }

            return $installed['versions'][$packageName]['version'];
        }

        throw new \OutOfBoundsException('Package "' . $packageName . '" is not
installed');
    }

    /**
```

```
     * @param  string      $packageName
     * @return string|null If the package is being replaced or provided but is not
really installed, null will be returned as version, use satisfies or getVersionRanges
if you need to know if a given version is present
     */
    public static function getPrettyVersion($packageName)
    {
        foreach (self::getInstalled() as $installed) {
            if (!isset($installed['versions'][$packageName])) {
                continue;
            }

            if (!isset($installed['versions'][$packageName]['pretty_version'])) {
                return null;
            }

            return $installed['versions'][$packageName]['pretty_version'];
        }

        throw new \OutOfBoundsException('Package "' . $packageName . '" is not
installed');
    }

    /**
     * @param  string      $packageName
     * @return string|null If the package is being replaced or provided but is not
really installed, null will be returned as reference
     */
    public static function getReference($packageName)
    {
        foreach (self::getInstalled() as $installed) {
            if (!isset($installed['versions'][$packageName])) {
                continue;
            }

            if (!isset($installed['versions'][$packageName]['reference'])) {
                return null;
            }

            return $installed['versions'][$packageName]['reference'];
        }

        throw new \OutOfBoundsException('Package "' . $packageName . '" is not
installed');
    }

    /**
     * @param  string      $packageName
     * @return string|null If the package is being replaced or provided but is not
really installed, null will be returned as install path. Packages of type metapackages
also have a null install path.
     */
    public static function getInstallPath($packageName)
    {
        foreach (self::getInstalled() as $installed) {
            if (!isset($installed['versions'][$packageName])) {
                continue;
            }

            return isset($installed['versions'][$packageName]['install_path']) ?
$installed['versions'][$packageName]['install_path'] : null;
        }
```

```
        throw new \OutOfBoundsException('Package "' . $packageName . '" is not
installed');
    }

    /**
     * @return array
     * @psalm-return array{name: string, pretty_version: string, version: string,
reference: string|null, type: string, install_path: string, aliases: string[], dev:
bool}
     */
    public static function getRootPackage()
    {
        $installed = self::getInstalled();

        return $installed[0]['root'];
    }

    /**
     * Returns the raw installed.php data for custom implementations
     *
     * @deprecated Use getAllRawData() instead which returns all datasets for all
autoloaders present in the process. getRawData only returns the first dataset loaded,
which may not be what you expect.
     * @return array[]
     * @psalm-return array{root: array{name: string, pretty_version: string, version:
string, reference: string|null, type: string, install_path: string, aliases: string[],
dev: bool}, versions: array<string, array{pretty_version?: string, version?: string,
reference?: string|null, type?: string, install_path?: string, aliases?: string[],
dev_requirement: bool, replaced?: string[], provided?: string[]}>}
     */
    public static function getRawData()
    {
        @trigger_error('getRawData only returns the first dataset loaded, which may
not be what you expect. Use getAllRawData() instead which returns all datasets for all
autoloaders present in the process.', E_USER_DEPRECATED);

        if (null === self::$installed) {
            // only require the installed.php file if this file is loaded from its
dumped location,
            // and not from its source location in the composer/composer package, see
https://github.com/composer/composer/issues/9937
            if (substr(__DIR__, -8, 1) !== 'C') {
                self::$installed = include __DIR__ . '/installed.php';
            } else {
                self::$installed = array();
            }
        }

        return self::$installed;
    }

    /**
     * Returns the raw data of all installed.php which are currently loaded for custom
implementations
     *
     * @return array[]
     * @psalm-return list<array{root: array{name: string, pretty_version: string,
version: string, reference: string|null, type: string, install_path: string, aliases:
string[], dev: bool}, versions: array<string, array{pretty_version?: string, version?:
string, reference?: string|null, type?: string, install_path?: string, aliases?:
string[], dev_requirement: bool, replaced?: string[], provided?: string[]}>}>
     */
    public static function getAllRawData()
```

```
    {
        return self::getInstalled();
    }

    /**
     * Lets you reload the static array from another file
     *
     * This is only useful for complex integrations in which a project needs to use
     * this class but then also needs to execute another project's autoloader in
process,
     * and wants to ensure both projects have access to their version of
installed.php.
     *
     * A typical case would be PHPUnit, where it would need to make sure it reads all
     * the data it needs from this class, then call reload() with
     * `require $CWD/vendor/composer/installed.php` (or similar) as input to make sure
     * the project in which it runs can then also use this class safely, without
     * interference between PHPUnit's dependencies and the project's dependencies.
     *
     * @param  array[] $data A vendor/composer/installed.php data set
     * @return void
     *
     * @psalm-param array{root: array{name: string, pretty_version: string, version:
string, reference: string|null, type: string, install_path: string, aliases: string[],
dev: bool}, versions: array<string, array{pretty_version?: string, version?: string,
reference?: string|null, type?: string, install_path?: string, aliases?: string[],
dev_requirement: bool, replaced?: string[], provided?: string[]}>} $data
     */
    public static function reload($data)
    {
        self::$installed = $data;
        self::$installedByVendor = array();
    }

    /**
     * @return array[]
     * @psalm-return list<array{root: array{name: string, pretty_version: string,
version: string, reference: string|null, type: string, install_path: string, aliases:
string[], dev: bool}, versions: array<string, array{pretty_version?: string, version?:
string, reference?: string|null, type?: string, install_path?: string, aliases?:
string[], dev_requirement: bool, replaced?: string[], provided?: string[]}>}>
     */
    private static function getInstalled()
    {
        if (null === self::$canGetVendors) {
            self::$canGetVendors = method_exists('Composer\Autoload\ClassLoader',
'getRegisteredLoaders');
        }

        $installed = array();

        if (self::$canGetVendors) {
            foreach (ClassLoader::getRegisteredLoaders() as $vendorDir => $loader) {
                if (isset(self::$installedByVendor[$vendorDir])) {
                    $installed[] = self::$installedByVendor[$vendorDir];
                } elseif (is_file($vendorDir.'/composer/installed.php')) {
                    $installed[] = self::$installedByVendor[$vendorDir] = require
$vendorDir.'/composer/installed.php';
                    if (null === self::$installed && strtr($vendorDir.'/composer',
'\\', '/') === strtr(__DIR__, '\\', '/')) {
                        self::$installed = $installed[count($installed) - 1];
                    }
                }
```

```
            }
        }

        if (null === self::$installed) {
            // only require the installed.php file if this file is loaded from its
dumped location,
            // and not from its source location in the composer/composer package, see
https://github.com/composer/composer/issues/9937
            if (substr(__DIR__, -8, 1) !== 'C') {
                self::$installed = require __DIR__ . '/installed.php';
            } else {
                self::$installed = array();
            }
        }
        $installed[] = self::$installed;

        return $installed;
    }
}
```

api/vendor/composer/LICENSE

```
Copyright (c) Nils Adermann, Jordi Boggiano

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is furnished
to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

api/vendor/composer/autoload_classmap.php

```php
<?php

// autoload_classmap.php @generated by Composer

$vendorDir = dirname(__DIR__);
$baseDir = dirname($vendorDir);

return array(
    'Composer\\InstalledVersions' => $vendorDir . '/composer/InstalledVersions.php',
);
```

api/vendor/composer/autoload_namespaces.php

```php
<?php

// autoload_namespaces.php @generated by Composer

$vendorDir = dirname(__DIR__);
$baseDir = dirname($vendorDir);

return array(
);
```

api/vendor/composer/autoload_psr4.php

```php
<?php

// autoload_psr4.php @generated by Composer

$vendorDir = dirname(__DIR__);
$baseDir = dirname($vendorDir);

return array(
    'Fhtechnikum\\Uebung34\\' => array($baseDir . '/src'),
);
```

api/vendor/composer/autoload_real.php

```php
<?php

// autoload_real.php @generated by Composer

class ComposerAutoloaderInit1fd71056277afaa2ee52250b805bde15
{
    private static $loader;

    public static function loadClassLoader($class)
    {
        if ('Composer\Autoload\ClassLoader' === $class) {
            require __DIR__ . '/ClassLoader.php';
        }
    }

    /**
     * @return \Composer\Autoload\ClassLoader
     */
    public static function getLoader()
    {
```

```
        if (null !== self::$loader) {
            return self::$loader;
        }

        spl_autoload_register(array('ComposerAutoloaderInit1fd71056277afaa2ee52250b805bc
'loadClassLoader'), true, true);
        self::$loader = $loader = new \Composer\Autoload\ClassLoader(\dirname(__DIR__));
        spl_autoload_unregister(array('ComposerAutoloaderInit1fd71056277afaa2ee52250b805
'loadClassLoader'));

        require __DIR__ . '/autoload_static.php';

call_user_func(\Composer\Autoload\ComposerStaticInit1fd71056277afaa2ee52250b805bde15::ge

        $loader->register(true);

        return $loader;
    }
}
```

api/vendor/composer/autoload_static.php

```
<?php

// autoload_static.php @generated by Composer

namespace Composer\Autoload;

class ComposerStaticInit1fd71056277afaa2ee52250b805bde15
{
    public static $prefixLengthsPsr4 = array (
        'F' =>
        array (
            'Fhtechnikum\\Uebung34\\' => 21,
        ),
    );

    public static $prefixDirsPsr4 = array (
        'Fhtechnikum\\Uebung34\\' =>
        array (
            0 => __DIR__ . '/../..' . '/src',
        ),
    );

    public static $classMap = array (
        'Composer\\InstalledVersions' => __DIR__ . '/..' .
'/composer/InstalledVersions.php',
    );

    public static function getInitializer(ClassLoader $loader)
    {
        return \Closure::bind(function () use ($loader) {
            $loader->prefixLengthsPsr4 =
ComposerStaticInit1fd71056277afaa2ee52250b805bde15::$prefixLengthsPsr4;
            $loader->prefixDirsPsr4 =
ComposerStaticInit1fd71056277afaa2ee52250b805bde15::$prefixDirsPsr4;
```

```
            $loader->classMap = ComposerStaticInit1fd71056277afaa2ee52250b805bde15::$classMap;

        }, null, ClassLoader::class);
    }
}
```

api/vendor/composer/installed.json

```
{
    "packages": [],
    "dev": true,
    "dev-package-names": []
}
```

api/vendor/composer/installed.php

```
<?php return array(
    'root' => array(
        'name' => 'fhtechnikum/uebung3_4',
        'pretty_version' => '1.0.0+no-version-set',
        'version' => '1.0.0.0',
        'reference' => NULL,
        'type' => 'library',
        'install_path' => __DIR__ . '/../../',
        'aliases' => array(),
        'dev' => true,
    ),
    'versions' => array(
        'fhtechnikum/uebung3_4' => array(
            'pretty_version' => '1.0.0+no-version-set',
            'version' => '1.0.0.0',
            'reference' => NULL,
            'type' => 'library',
            'install_path' => __DIR__ . '/../../',
            'aliases' => array(),
            'dev_requirement' => false,
        ),
    ),
);
```