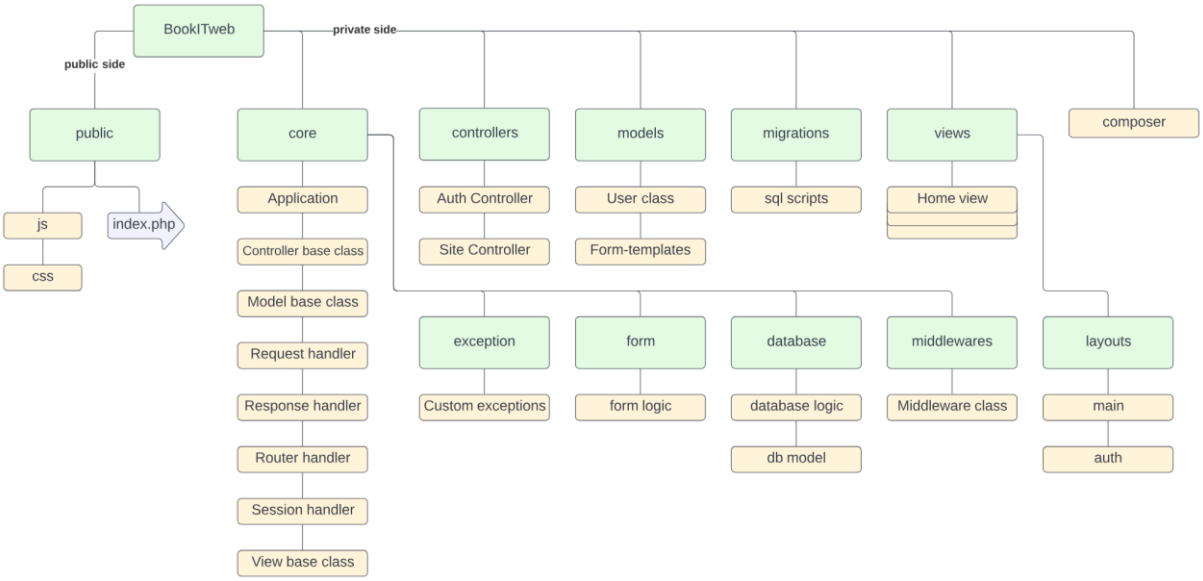
# Modul 5

*Glenn Joakim Bakklund | Tom André Slåen Myre | Prosjektgruppe 6*

## Oppgave 5_1: Plan for filstruktur og klassebibliotek

### Filstruktur

Vi har som plan å strukturere fellesprosjektet vårt som et MVC prosjekt. Dette lar oss fordele kontroll-logikk og oppgaver basert på ansvar, og gir oss et ryddig og definert prosjekt. Det lar oss også skille mellom hva som presenteres og gjøres tilgjengelig for brukere. Figur 1 viser vår plan for filstrukturen til prosjektet:




*Figur 1: Filstruktur for fellesprosjekt i PHP.*

Her er forklaring på de forskjellige mappene (alle grønne boksene i figur 1):

- **public**/: Inneholder offentlige ressurser som css og js, i tillegg til index.php som fungerer som eneste inngangspunkt for resten av systemet.
    - **css**/: Stylesheet for utforming av systemet.
    - **js**/: JavaScript-filer for spesielle interaksjons-handlinger blant annet.
- **core**/: Inneholder kjernelogikk for systemet, og har i oppgave å bygge sammen hva som sendes ut til brukere basert på hva som blir forespurt.
    - **exception**/: Inneholder feilmeldinger.
    - **form**/: Inneholder logikk for skjemaer.
    - **database**/: Inneholder logikk for databaser.
    - **middlewares**/: Inneholder mellomstegs-logikk som et ekstra lag mellom klient og server. Eksempler er prosesser som logging og autentisering.
- **controllers**/: Inneholder kontrollere med logikk for hvordan sidene skal struktureres.
- **models**/: Inneholder klasser for informasjons-håndtering.
- **migrations**/: Inneholder sql-script for migrering av database.
- **views**/: Html og side-innhold for hver side.
    - **layouts**/: Inneholder hovedstruktur for sidene, med felles head-element til sidene.

## Klasse bibliotek

Core-mappen:

- **Application**: Bygger opp og binder sammen alle nødvendige klasser for systemet, og fungerer som knutepunktet/hovedinngang til kjernelogikken av systemet.
- **Controller base class**: Inneholder grunn-funksjoner som arves av alle kontrollere.
- **Model base class**: Inneholder grunn-logikk og funksjoner som arves av alle modeller.
- **Request handler**: Inneholder logikk for håndtering av forespørsler fra klienten.
- **Response handler**: Inneholder logikk for å sende server-responser til klient. Eksempler er redirigering og http-koder.
- **Router handler**: Ansvarlig for håndtering av klient-forespørsler for å dirigere til korrekt kontroller. Har også ansvar for kjøring av kontrollerens registrerte mellomvarer.
- **Session handler**: Inneholder logikk for styring av sessions for innloggede brukere.
- **View base class**: Inneholder grunn-logikk og funksjoner som arves av alle views.

Controller-mappen:

- **Auth Controller**: Har ansvar for begrensede sider, og sider for autorisering. Eksempel er login.
- **Site Controller**: Har ansvar for offentlige sider.

Model-mappen:

- **User Class**: Inneholder logikk og funksjoner for håndtering av bruker-relaterte operasjoner.

## Oppgave 5_2: Kommentering

Fellesprosjektet har på dette tidspunktet en god del filer og linjer med kode, så vi har valgt å ta noen utdrag fra koden til for å vise eksempler på vår kommentering.

```php
use app\core\Application;
use app\controllers\SiteController;
use app\controllers\AuthController;

//getting dependencies
require_once __DIR__ . '/../vendor/autoload.php';
$dotenv = Dotenv\Dotenv::createImmutable(dirname(__DIR__));
$dotenv->load();

//configuring the application
$config = [
    'userClass' => \app\models\User::class,
    'db' => [
        'dsn' => $_ENV['DB_DSN'],
        'user' => $_ENV['DB_USER'],
        'password' => $_ENV['DB_PASSWORD']
    ]
];

//creating the application with the config
$app = new Application(dirname(__DIR__), $config);


//creating the routes
//site routes
$app->router->get('/', [SiteController::class, 'home']);
$app->router->get('/contact', [SiteController::class, 'contact']);
$app->router->post('/contact', [SiteController::class, 'contact']);

//auth routes
$app->router->get('/login', [AuthController::class, 'login']);
$app->router->post('/login', [AuthController::class, 'login']);
$app->router->get('/register', [AuthController::class, 'register']);
$app->router->post('/register', [AuthController::class, 'register']);
$app->router->get('/logout', [AuthController::class, 'logout']);
$app->router->get('/profile', [AuthController::class, 'profile']);
$app->router->get('/admin', [AuthController::class, 'admin']);

//starting the application
$app->run();
```

*Figur 2: Bilde som viser kommentering i index.php i fellesprosjektet.*

```php
use app\core\exeption\NotFoundExeption;
/**
 * Class Router
 *
 * Responsible for routing user requests to the correct controller and action,
 * based on the request method and path. Routes must be predifined to be used.
 *
 * @version 1.0
 * @author GlennJoakimB <89195051+GlennJoakimB@users.noreply.github.com>
 * @package app\core
 */
2 references | 0 implementations
class Router
{
    4 references
    public Request $request;
    2 references
    public Response $response;
    3 references
    protected array $routes = [];

    /**
     * Router constructor
     *
     * @param \app\core\Request $request
     * @param \app\core\Response $response
     */
    0 references | 0 overrides
    public function __construct(Request $request, Response $response)
    {
        $this->request = $request;
        $this->response = $response;
    }
```

*Figur 3: Utklipp som viser kommentering til Router-klassen i fellesprosjektet.*

```php

namespace app\core
{
    /**
     * Model short summary.
     *
     * Model is the base class for all models, it provides the basic
     * functionality for validation and error handling.
     *
     * @version 1.0
     * @author Trivinyx <tom.a.s.myre@gmail.com>
     * @package app\core
     */
    32 references | 6 implementations
    abstract class Model
    {

        13 references
        public const RULE_REQUIRED = 'required';
        6 references
        public const RULE_EMAIL = 'email';
        4 references
        public const RULE_MIN = 'min';
        4 references
        public const RULE_MAX = 'max';
        4 references
        public const RULE_MATCH = 'match';
        4 references
        public const RULE_UNIQUE = 'unique';

```

*Figur 4: Utklipp som viser kommentering til Model-klassen i fellesprosjektet*

```php
    /**
     * Render the view with given paramaters
     *
     * @param mixed $view
     * @param mixed $params
     * @return array|string
     */
    7 references | 0 overrides
    public function render($view, $params = []): string
    {
        return Application::$app->view->renderView($view, $params);
    }

    /**
     * Register middleware
     *
     * @param \app\core\BaseMiddleware $middleware
     */
    1 reference | 0 overrides
    public function registerMiddleware(BaseMiddleware $middleware){
        $this->middlewares[] = $middleware;
    }

    /**
     * Get all middlewares
     *
     * @return \app\core\BaseMiddleware[]
     */
    1 reference | 0 overrides
    public function getMiddlewares(): array
    {
        return $this->middlewares;
    }
```

*Figur 5: Utklipp som viser kommentering fra metoder i Controller-klassen i fellesprosjektet.*