Úvod

Cílem práce je vytvoření webové aplikace pro dispečink taxislužby, která by měla zcela nahradit papírovou evidenci a tím výrazně zefektivnit získání přehledů o objednávkách, o docházce řidičů, vozidlech a vazbách mezi nimi. Papírová evidence je časově náročná na vyhledávání a tvorbu přehledů objednávek pro řidiče za zadané období.

Zadávání veškerých dat obstarává dispečer, komunikující s řidiči vysílačkou a objednávky přijímá telefonicky v nepřetržitém směnném provozu.

Aplikace usnadňuje a zpřesňuje práci dispečera při velkém počtu řidičů k dispozici a vyřizování velkého množství příchozích objednávek v období poptávkové špičky, kdy dispečer je snadno zahltitelný a může dělat chyby.

Uživatelské rozhraní dispečerovi poskytuje přehled o stavu řidičů a stavu objednávek a řadí je do fronty k vyřízení. Výsledná aplikace by měla tuto agendu zcela nahradit.

Struktura práce začíná úvodem do řešené problematiky dispečinku. Následuje seznámení s použitými technologiemi a jejich výhodami. Dále pak vlastní řešení aplikace pro dané problémy. Poskytuji jednoduchý návod na použití pro uživatele. Závěrem bude shrnutí projektu a jeho vyhodnocení.

Motivace

Dispečerova práce v taxislužbě spočívá v komunikaci se zákazníky pomocí telefonu, či mobilní aplikace a s řidiči vozidel taxi pomocí vysílačky. Obvykle je přítomen dispečer v nepřetržitém provozu na směny.

Objednávky včasné i okamžité zapisuje do papírových sešitů. Po jejich vyřízení, tím je myšleno předání konkrétnímu řidiči k vyřízení zapisuje řidiče a vozidlo, kterým byla objednávka vyřízena.

Druhou agendou je docházka řidičů do práce. Vždy jsou k dispozici řidiči pracující na směny. Vedle nich jsou k dispozici řidiči, kteří nejsou na své směně, ale pouze čekají na zákazníky na stanovištích a nahlásili se dispečerovi jako záloha.

Řidiči v taxislužbě jsou koncesovaní živnostníci a od dispečinku si přidělené objednávky kupují. Proto je evidence všech pohybů nutná, aby mohl každý řidič jednou za měsíc dostat výpis jemu přidělených objednávek k vyrovnání. Tvorba přehledů a součtů je v papírové formě velmi zdlouhavá práce, kterou bude aplikace jednoduše automatizovat.

V poptávkové špičce dochází často k chybám na pravidlech přidělování objednávek, například může být čekající klient zapomenut, nebo řidič vyslán někam, kde už byla objednávka vyřízena.

Jako webová aplikace může být používána z libovolného místa a tím umožní vykonávat dispečerskou práci z domu za pomoci přenosné vysílačky, mobilního telefonu a zařízením s webovým prohlížečem, například tabletem. Tato skutečnost by měla vést ke zrušení pronajaté kanceláře a přechodu na práci z domu pro osoby se sníženou schopností pohybu a orientace.

Řešená problematika

Cílem práce je návrh a implementace webové aplikace pokrývající písemnou agendu dispečerské práce. Seznámení se a použití níže uvedených technologií pro tvorbu webové aplikace typu Single page aplikace. Implementovanou aplikaci otestovat na zkušební sadě dat v databázi.

Dispečink taxislužby je obsluhován v nepřetržitém směnném provozu a přijímá telefonické objednávky. Objednávky jsou řazeny do fronty podle jejich času k vyřízení. Ty okamžité jsou řazeny podle času jejich vzniku. Pokud se sejde v jeden čas okamžitá a včasná objednávka, pak včasná má vyšší prioritu.

Řidiči jsou řazeni do fronty pro další přicházející objednávku, ve které mají přednost ti na své směně a pak ostatní k dispozici. Dále tvorba fronty musí zohledňovat stav řidiče, zda není mimo město, nemá pauzu, nebo je nepřítomen ve voze. Řidičům mimo směnu přiděluje dispečer objednávky pouze pokud ti ve směně jsou obsazeni a nemohou příchozí objednávku obsloužit.

Uživatelské rozhraní by mělo tyto dvě fronty zobrazit a tím poskytnout dispečerovi možnost přidělovat objednávky bez chyb a spravedlivě podle výše zmíněných pravidel. Pro potřeby taxislužby je evidována i docházka dispečerů spolu s docházkou řidičů. Dále logování stavu řidičů během přítomnosti v práci pro přehled průběhu směny.

Použité technologie

PHP Framework Slim

Micro php mvc framework pro serverovou část aplikace, který je určen pro rychlou tvorbu malých webových aplikací, nebo API. Přijímá http dotazy, provede určité rutinní akce, obvykle práce s databází a vrátí http odpověď. Díky malému množství zdrojového kódu je v porovnání s klasickými PHP frameworky velmi odlehčený a rychlý. Poskytuje minimální sadu nástrojů pro serverovou část klient-server webové aplikace. Tento framework jsem porovnal s jinými volně dostupnými a zjistil jsem, že pro mou aplikaci nenabízí význačně vhodnější možnosti. [1]

Slim Framework používá architekturu REST, tudíž podporuje všechny http metody (GET, POST, PUT, DELETE). Pomocí middleware vrstev lze aplikovat různě obecná pravidla a lze tak přesně upravit každou http odpověď.

Databáze MySQL

MySQL je jedna z nejpoužívanějších a nejoblíbenější open source relačních databází na světě. Databáze MySQL se díky osvědčené výkonnosti, spolehlivosti a snadnému používání stala volbou číslo jedna pro webové aplikace. [2]

Od verze 5.1 dostupné od roku 2008 se již řadí mezi plnohodnotné databázové systémy díky přidání podpory transakcí, triggerů a pohledů. Pro svůj výkon a především proto, že se jedná open-source software, má vysoký podíl na v současné době používaných databázích.

React

Jedná se o javascriptovou knihovnu pro tvorbu interaktivních uživatelských rozhraní, převážně single page aplikací. Umožňuje navrhnout interaktivní uživatelské rozhraní a vykreslovat komponenty podle vnitřního stavu aplikace v reálném čase.

Každá komponenta musí implementovat metodu **render()**, která na základě vnitřního stavu komponenty vrací to, co se má zobrazit. Dále je důležitý objekt **props,** zkratka slova properties, který je povinným parametrem pro konstruktor každé komponenty.

Díky logice založené na jazyce JavaScript bývá obvykle datový model na straně klienta udržován v pomocných objektech, které jsou předávány do komponent jako parametry props. V těchto modelových objektech se volají procedury na načtených datech, či jejich získávání a posílání na serverové api.

Komponenty lze skládat libovolně, obvykle se každá komponenta skládá z menších a každá má odpovědnost za vykreslení konkrétní části DOM. Po vytvoření celé struktury DOM, se nadále aktualizují pouze ta místa, která hodnotou závisí na props objektu. [3]

Na ukázce kódu v obrázku č. 1 vidíme jednoduchou komponentu HelloText, která vrátí nadpis s textem, který je předán pomocí props. Dále zastřešující komponentu *App2*, která jí používá a předává jí hodnotu „Dobrý den komponento“. Výsledkem tohoto kódu je nadpis s vloženým textem.

```
export interface MyProps {
    helloText: string;
}

export class HelloText extends React.Component<MyProps>{
    render() {
        return (
            <h1>{this.props.helloText}</h1>
        );
    }
}

export class App2 extends React.Component {
    render() {
        return (
            <div className="mainContainer">
                <HelloText helloText={"Dobry den komponento"} />
            </div>
        );
    }
}

ReactDOM.render(<App2 />, document.getElementById('root') as HTMLElement);
```

TypeScript

Jedná se o nadstavbu jazyka JavaScript. Nabízí statickou typovou kontrolu pro primitiva, třídy, rozhraní, návratové typy metod apod. Jeho použití výrazně ulehčuje vývoj. Na úrovni vývojového prostředí ihned signalizuje typovou nekompatibilitu a možné chyby, nabízí možnost navigace k definici. Dále usnadňuje refaktorování kódu. [5]

Pokud voláme například REST služby, které nám vracejí data ve formátu JSON, neznáme jejich schéma. Poté co s těmito daty pracujeme, musíme mít jejich strukturu v hlavě. Definujeme-li rozhraní v TypeScriptu, které popisuje, jaké má objekt vlastnosti a jakého jsou typu, značně si ulehčíme práci, ladění a zamezíme mnoha chybám.

Na obrázku č. 2 použití spočívá v definování typu za dvojtečkou u atributů. U komponent lze definovat rozhraní pro props ve špičatých závorkách.

Parcel

Parcel lze použít bez další konfigurace na sestavení aplikace do jednoho souboru index.js, který je jako jediný importován do kořenového souboru index.html. Šetří čas při vývoji aplikace, kdy sám sleduje změny ve zdrojových souborech a aktualizuje sestavenou webovou aplikaci a soubor index.js. Alternativou může být například webpack, či browserify, které jsou vhodné pro velké projekty. [6]

```
C:\xampp\htdocs\TaxiDisp\src>parcel watch --out-dir ./public/scripts ./taxi-ui/s
rc/index.tsx
?  Built in 2.99s.
```

MobX

MobX je javascriptová knihovna pro management stavu mezi komponentami a modelovými objekty. Nejedná se o stavový kontejner, jako například knihovna Redux. Používá se jako tzv. wrapper nad existujícími komponentami a modely a poskytuje propsání aktuálních hodnot v modelech do komponent a do přímo do zobrazených hodnot v metodě render konkrétní komponenty. MobX je nejčastěji používána ve spojení s knihovnou React. [4]

Použití spočívá v anotaci **@observer** nad deklarací komponenty, čímž říkáme, že komponenta bude pozorovatelem všech svých primitivních i objektových parametrů v objektu props, které jsou označeny anotací **@observable** při deklaraci uvnitř modelových objektů, čímž jsou dekorovány na objekt tohoto typu. Změny takto anotovaných atributů budou ihned viditelné všude, kde jsou čteny libovolnou komponentou anotovanou jako observer.

Dále jsou důležitou částí **@computed** hodnoty, které lze odvodit ze stávajícího stavu, nebo jiných vypočtených hodnot. Chovají se podobně jako vzorce v tabulkách Excel. Na základě definované operace vrací hodnotu podle aktuálního stavu použitých atributů k výpočtu a poskytují vždy aktuální hodnotu.

Na obrázku č. 3 uvádím jednoduchý model, který při zavolání metody inkrementuje atributy *n1* a *n2* a poskytuje vypočtenou hodnotu, která je jejich součtem. Zastřešující komponenta *App2* čte z modelu vypočtenou hodnotu a celý model předává níže do vnořené komponenty, která volá akce definované na modelu, v tomto případě inkrementování atributů. Vždy je zobrazena aktuální hodnota.

```
export class MyModel {
    @observable n1: number;
    @observable n2: number;
    constructor() {
        this.n1 = this.n2 = 0;
    }
    @computed get sumN1N2() {
        return (this.n1 + this.n2);
    }
    @action.bound
    addOne() {
        this.n1 = this.n1 + 1;
        this.n2 = this.n2 + 1;
    }
}
const model1 = new MyModel();
export interface MyProps {
    helloText: string;
    model: MyModel;
}
@observer
export class HelloText extends React.Component<MyProps>{
    render() {
        return (
            <div>
                <h1>{this.props.helloText}</h1>
                <button onClick={() => this.props.model.addOne()}>inc</button>
            </div>
        );
    }
}
@observer
export class App2 extends React.Component {
    render() {
        return (
            <div className="mainContainer">
                <HelloText model={model1} helloText={"Dobry den komponento"} />
                <span>n1 + n2 = {model1.sumN1N2}</span>
            </div>
        );
    }
}
ReactDOM.render(<App2 />, document.getElementById('root') as HTMLElement);
```

Bootstrap

Jedná se o open-source CSS framework, neboli sadu hotových stylů, které lze snadno použít a ušetřit čas a práci při stylování uživatelského rozhraní webové aplikace oproti stylování od úplného základu. Nabízí mřížku, výchozí typografii, stylování pro formulářové elementy. Tvorba vizuální podoby se tak stává jednoduchým sestavováním vizuálních komponent. Obvyklá praxe je, použití Bootstrapu jako dobrého základu pro vizuální podobu a rozvržení a dále přidáním vlastních stylů se upravuje vzhled podle přesných požadavků a specifik aplikace. Výrazným přínosem tohoto css frameworku pro webové aplikace je vyřešená responzivita uživatelského rozhraní pro malá zařízení [7]

Single page aplikace

Single page application (někdy také one page application) je typ webové aplikace, která používá server pouze jako zdroj a úložiště dat. Data jsou potom kompletně vykreslována JavaScriptem.

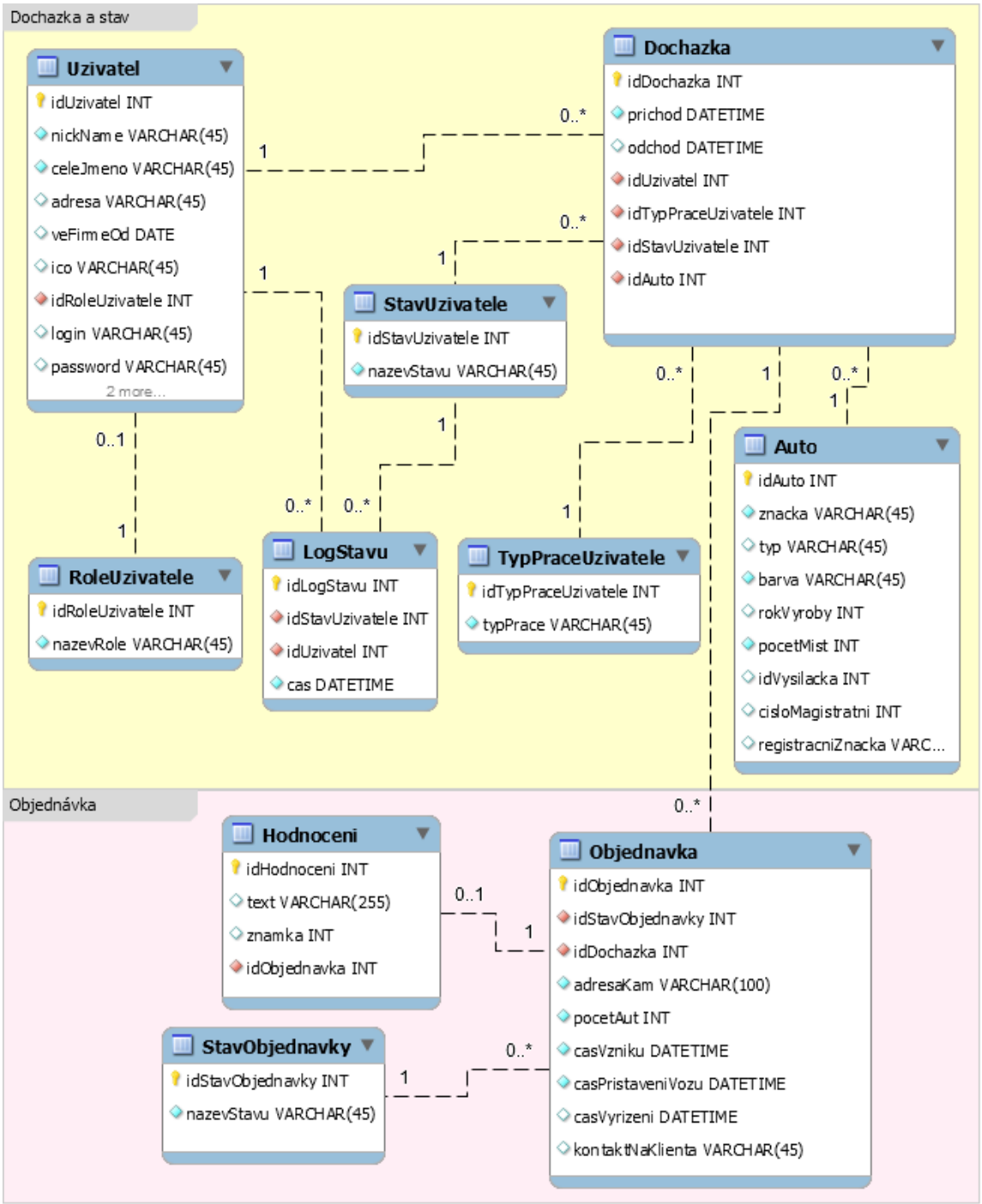
Po prvním příchodu na stránku dochází k stáhnutí potřebných javascriptů a k jejich spuštění. Dojde k asynchronnímu načtení potřebných dat ze serverového api do objektových struktur na straně klienta.

Hlavní výhodou SPA je rychlost, protože je překreslována pouze ta část, která se mění. Serverová část neposkytuje html kód, ale pouze data a to pouze ta data, která jsou aktuálně potřeba. Obvykle ve formátu JSON. Při relativně objemných datových strukturách se používá lazy-loading, kdy se nahrává nejdůležitější obsah, který se zobrazí nejdříve a na pozadí se donahrávají zbývající data.

Nevýhodou je delší čas prvního načtení stránky, která stahuje potřebné skripty, poté teprve odchází k načtení potřebných dat a jejich zobrazování. Podmínkou k použití je funkční JavaScript v prohlížeči na klientském zařízení. [8]

Vlastní řešení

Datový model



Uživatel

Jedná se o dispečery a řidiče, kterým bude evidována docházka. Obsahuje pouze sadu základních atributů.

RoleUzivatele

Každý uživatel má roli v rámci firmy, buď je řidičem, nebo dispečerem. Pokud by se firma rozhodla evidovat docházku u jiné pozice, jednoduše ji vytvoří jako nový řádek této tabulky. Na základě této informace může být rozhodnuto o druhu uživatelského rozhraní, kde uživatelé mohou získat přehledy odpracovaných dob.

StavUzivatele

Pokud je uživatel přihlášen v práci, má nějaký stav podle charakteru vykonávané práce taxislužby, například.: volný, obsazený, tankování, soukromé apod.

LogStavu

Při změně stavu uživatele na jiný stav, se uloží ten předchozí, včetně času změny. Pro pozdější možnost přehledu charakteru vytíženosti směny, či jiného časového období.

TypPraceUzivatele

Nyní jsou tři typy práce: ve službě, mimo směnu a pohotovost. Na základě tohoto atributu docházky je sestavována fronta řidičů na přicházející objednávky. Řidiči pracující na své směně mají vyšší prioritu, než řidiči mimo směnu k dispozici. Nejnižší prioritu mají pohotovostní řidiči, kteří jsou volání v případě neodkladné výpomoci, obvykle v tomto stavu jsou doma a tento typ docházky sami nahlásili, přičemž obvykle po jejich zavolání přechází do stavu mimo směnu.

Dochazka

Hlavní tabulka pro evidenci docházky, podle níž je odvozeno, kdo je přítomen v práci, jaký je typ jeho práce, kterým autem je v práci a jaký má stav. Pokud je přítomen, má pouze příchod vyplněný na daném řádku tabulky.

Auto

Jedná se o vozidla provozovaná pod vlajkou konkrétní taxislužby a jejich základní atributy. Značka a barva bývá sdělena klientovi k rozpoznání vozidla, na které čeká.

Objednavka

Po docházce druhá hlavní entita celé aplikace. Objednávka vzniká na základě akcí dispečera ve webovém rozhraní. Obsahuje sadu základních atributů, odkaz na svůj stav a odkaz na konkrétní docházku, z čehož lze zjistit vozidlo a řidiče, který objednávku obsluhoval.

StavObjednavky

Každá objednávka má stav určený na základě jejího postupu při obsluhování: přidělena, obsluhována, zrušena, vyřízena úspěšně apod.

Hodnoceni

S touto tabulkou zatím není pracováno v rámci webové aplikace, je zde pro budoucí použití v rámci udělování hodnocení od klientů pomocí mobilní aplikace. Bude rozšířena o hvězdičkové známkování.

Serverová část ve frameworku Slim

Modelové objekty

Jedná se o immutable objekty, tudíž nabízí pouze **get()** metody a nastavení atributů probíhá pouze pomocí parametrů v konstruktoru.

Pro každou entitu v databázi byla vytvořena modelová třída, která obsahuje primitivní a objektové atributy. Objektové atributy, čili jiné entity, jsou doplněny na základě jejich cizích klíčů v dané entitě.

Modelové třídy zároveň používám jako DTO (data transfer object), tím že všechny implementují rozhraní JsonSerializable. Pokud by bylo potřeba specifický objekt, který není databázovou entitou, jednoduše ho přidám a pomocí servisní třídy naplním daty z databáze.

```php
namespace App\Models;

use App\Models\RoleUzivatele;

class Uzivatel implements \JsonSerializable {

    private $id;
    private $nickName;
    private $login;
    private $celeJmeno;
    private $role;

    function __construct($nickName, $login, $celeJmeno,
                         RoleUzivatele $role, $id = null) {
        $this->id = $id;
        $this->nickName = $nickName;
        $this->login = $login;
        $this->celeJmeno = $celeJmeno;
        $this->role = $role;
    }

    public function jsonSerialize() {
        return get_object_vars($this);
    }

    function getId() {
        return $this->id;
    }
```

Servisní objekty

Ke každému modelovému objektu používám servisní objekt, který je zodpovědný za komunikaci s databází a osazováním modelových tříd daty a jejich sestavováním do objektových struktur, pokud je potřeba za cizí klíč dosadit objekt jiné entity.

Každá metoda je volána jinou akcí z kontrolleru a poskytuje pro konkrétní akci potřebná data, která kontroller poskytuje v api. []

```php
namespace App\Services;

use App\Services\AService;
use App\Models\Uzivatel;
use PDO;
/**
 * UzivatelService - servisni objekt dodavajici
 * data z databaze kontroleru pomoci DTO
 * @author Jan Špecián
 */
class UzivatelService extends AService {

    public function getAllUzivatel() {
        $sql = "
        SELECT *
        FROM Uzivatel";
        $stmt = $this->container->db->prepare($sql);
        $stmt->execute();
        $results = $stmt->fetchAll();

        $uzivatele = [];
        foreach ($results as $obj) {
            $u = $this->assemblyDTO(($this->getUserById($obj['idUzivatel'])));
            $uzivatele[] = $u;
        }
        return $uzivatele;
    }
```

Routing

Ve Frameworku slim se jedná o jednoduché volání funkcí na instanci celé aplikace. Lze použít seskupování podobných dotazů. Podle příchozího dotazu je rozhodnuto jaký kontroller a jaká jeho metoda bude použita.

```
$app->get('/', HomeController::class . ':home');

$app->group('/Uzivatele', function() {
    $this->get('/{id}', UzivatelController::class . ':getUzivatelDetailById');
    $this->get('', UzivatelController::class . ':getAllUzivatel');
    $this->post('', UzivatelController::class . ':saveNewUzivatel');
    $this->map(['PUT', 'PATCH'], '/{id}', UzivatelController::class .
':updateUzivatel');
    $this->delete('/{id}', UzivatelController::class . ':deleteUzivatel');
});
```

Middleware

Každý framework používá middleware jiným způsobem. Zde se jedná o soustředné vrstvy nad každou skupinou metod, nebo nad všemi metodami. Každá další vrstva obalí již existující middleware vrstvy. Poslední přidaná vrstva je pak vykonávána jako první. V této aplikaci byla například použita jedna vrstva přes veškeré api pro nastavení Content-Type pro http odpovědi na typ application/json.

Dependency Container

V kontejneru se nachází PDO objekt pro spojení s mysql databází. Dále základní pohled, který vrací základní html stránku s vloženým javascriptem při zavolání HomeControlleru. Nechybí zde všechny servisní objekty k modelovým objektům, které používají kontrollery.

```
$container = $app->getContainer();
$container['db'] = function ($c) {
    $db = $c['settings']['db'];
    $pdo = new PDO('mysql:host=' . $db['host'] . ';dbname=' . $db['dbname'],
                   $db['user'], $db['pass']);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
    return $pdo;
};
```

Klientská část

- Hlavní komponenty
- Pomocné objekty
- Mobx a React Router

api call, react, mobx, asynchroní a synchronní části, dom,

gui ukázky

React

MobX

Návod k použití

- Spuštění
- Pravidla taxislužby a fronty

Závěr

- není přihlašování
- možno dodefinovat další možné přehledy
- rozšiřitelnost
- znovupoužitelnost i pro jiné dispečinky taxislužeb podobné velikosti

Shrnutí výsledků projektu a závěr (naznačení dalšího možného pokračování)

Zdroje

1. https://www.slimframework.com/docs/
2. Mysql web
3. https://reactjs.org/docs/rendering-elements.html
4. https://mobx.js.org/getting-started.html
5. https://www.zdrojak.cz/clanky/k-cemu-je-dobry-typescript/
6. Parcel web
7. http://getbootstrap.com/
8. http://jecas.cz/spa
9. https://www.toptal.com/php/maintain-slim-php-mvc-frameworks-with-a-layered-structure
10.

http://frontendinsights.com/connect-mobx-react-router/

https://mobx.js.org/best/store.html

https://tylermcginnis.com/courses/react-router/

Osnova

- Použité technologie
- Klient server webová aplikace
- Serverová část
    - Databáze, jednotlivé entity a jejich použití
    - Api
    - Dto
- Klientská část
    - Hlavní komponenty
    - Pomocné objekty
    - Mobx a React Router
- Návod k použití aplikace
    - Spuštění
    - Pravidla taxislužby a fronty

- Přehledy