Rapport:

Inlämningsuppgift u05-car-rental

Syfte

Uppgiften är att utveckla en applikation med hjälp av PHP, JavaScript och Twig som beskriver ett system för uthyrning av bilar. Applikation skall baseras på MVC-modellen och kommunicera via en router. Applikationens data skall lagras i en databas med hjälp av MySQL.

Jag har valt att avvika något från den struktur på uppgiften som framgått av den delade specifikationen. I stället för att ha en speciell sida för huvudmeny har jag valt att integrera denna på samtliga sidor. För mig framstår detta som betydligt mer användarvänligt då det undviker onödig navigering. Jag har också gett användaren möjlighet att ångra gjorda raderingar från kund- och biltabellerna. Tyvärr har jag inte haft tid att göra så att en sådan ändring även backar gjorda rensningar i uthyrningstabellen. För detta skulle samtliga ändrade rader behöva lagras tillfälligt för att kunna återställas. Det är inget stort problem men jag har inte gjort det.

Arkitektur

Applikationen baseras på en .htaccess fil som styr om all trafik till en och samma index.php fil. I denna index-fil skapas en instans av klassen Router som med hjälp av ett objekt av klassen Request analyserar den sökväg som angetts i URI:n. Routern bryter ner sökvägen i sina beståndsdelar vilka genom regex-mönstermatchning paras ihop med möjliga sökvägar lagrade i en separat fil. Genom att matcha sökvägen mot de fördefinierade vägarna styrs anropet till det Controller-objekt (kontrollanten) som ansvarar för denna typ av anrop och den aktuella metoden på kontrollanten anropas.

Den åberopade metoden på kontrollant-objektet ansvarar för eventuell applikationslogik och att, vid behov, kommunicera med datalagret för att därefter vidarebefordra data att presentera skärmen. Själva presentationen sker genom Twig. Kommunikationen med datalagret sker genom separata modeller som åberopas från kontrollanten.

Kontrollanterna kommunicerar, som sagt, inte direkt med databasen utan med modeller, klasser, som ansvarar för att i sin tur skriva, modifiera, ta bort och hämta data från databasen. Då de olika kontrollerna gör liknande anrop har applikationen en super-modell-klass som de andra modellerna ärver från. Det är denna superklass som ansvarar för kommunikationen med databasen medan underklasserna ansvarar för att fungera som ett interface gentemot kontrollerna. På så vis undviks repetitiv kod och genom underklasserna kan relativt komplicerade metodanrop döljas från kontrollerna.

Datalagrets modeller, vilka är egna klasser, motsvarar M (Model) i MVC-modellen. De presentationer som görs av Twig gentemot användaren motsvarar V (View) i MVC-modellen. Slutligen motsvarar kontrollanterna C (Controller).

Dokumentation

Applikationen är utförligt kommenterad vilket får tjäna som den primära dokumentationskällan. Här kommer en översiktlig genomgång av beståndsdelarna. Denna genomgång kommer att följa den befintliga mappstrukturen.
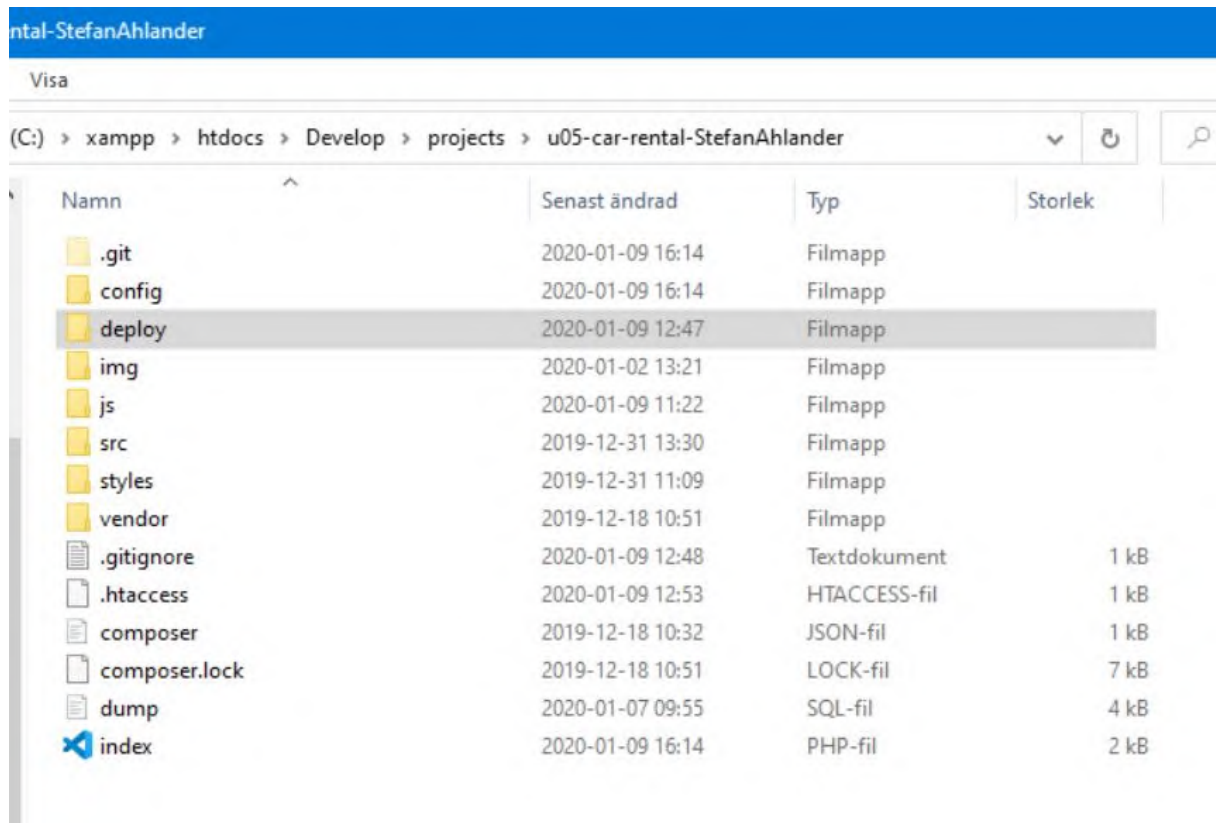
Config-mappen innehåller en config-fil med inloggningsuppgifter för databasen samt applikationens "bas-url". Bas-url används för att applikationen på min server inte ligger i rot-foldern utan i en under-katalog. I config-mappen finns även filen, routes.json, som innehåller applikationens sökvägar och som används av routern.

Deploy-mappen innehåller modifierade versioner av config-filen och routes.json-filen för den version av applikationen som ligger uppe på Internet. Dessa filer versions hanteras inte av säkerhetsskäl.

I img-mappen finns bara en bakgrundsbild som visas på startsidan.

Mappen js innehåller den Javascript fil som laddas in varje gång en ny vy visas. Javscript-koden används för att verifiera användarens input innan denna lagras i databasen.

I src mappen ligger den huvudsakliga applikationskoden och jag återkommer till denna nedan.

Styles-mappen innehåller en CSS-fil för att göra applikationen något trevligare att titta på.

Vendor mappen innehåller filer kopplade till Twig. Det är där som composer lagrar de filer eller moduler som installerats.
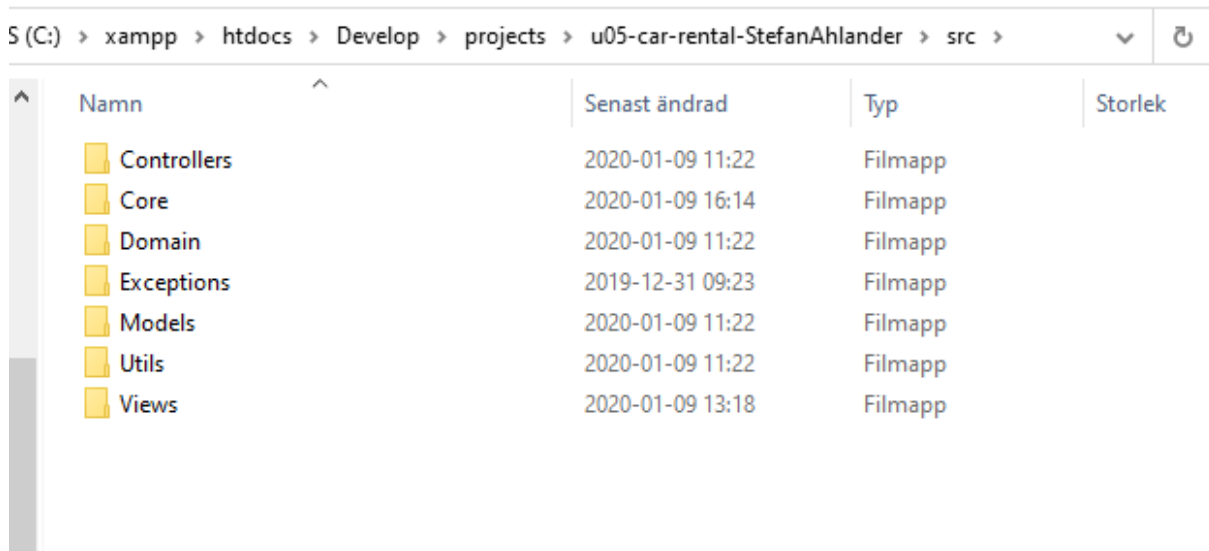
.gitignore filen används för att förhindra att "vendor"-mappen samt "deploy"-mappen versions hanteras.

Genom .htaccess filen styrs alla sidanrop om så att index.php filen körs oavsett sökväg.

Composer och composer.lock är filer osm används av Composer för att hantera insticksmoduler samt den autoloader som används för applikationen.

Dump.sql filen innehåller de kommandon som "bygger" den ursprungliga databasen som används av applikationen.

Slutligen är index.php applikationens hjärta, den fil som hanterar hela applikationen.

Src-mappen har flera mappar som innehåller de olika delarna av applikationen.

Controllers, Models och Views är ganska självförklarande.

Core-mappen innehåller filer som är gemensamma för applikationen. Där finns Config-klassen som används för att ta fram data ur config-filen. Där finns också en hjälp-klass, FilteredMap, som används för att "tvätta" information som lämnas av användaren innan denna läggs in i databasen. Detta för att försöka minska risken för angrepp genom så kallad SQL-injection. I mappen finns också klasserna Request och Router som ansvarar för att styra användaren rätt i applikationen genom att analysera de angivna sökvägarna.

Mappen domain innehåller klasser som beskriver de objekt som applikationen använder sig av. Bilar, kunder och uthyrningar. Det är dessa objekt som skapas, modifieras, tas bort och lagras i databasen.

Exceptions innehåller klasser för att kunna avgöra vilka typer av fel som inträffar när applikationen används.

I mappen Utils slutligen så ligger den klass, DependencyInjector, som används för att skicka runt applikationsberoenden utan att behöva lista separata beroenden för de olika delarna av applikationen.

Exempel på exekvering

När en användare navigerar till applikationen kommer denne att genom .htaccess filen dirigeras till filen index.php. När denna efterfrågas på servern kommer php-skriptet att köras.

I index-filen görs flera saker. Applikationen konfigureras och därefter analyseras den angivna sökvägen och genom routern åberopas "rätt" kod utifrån angiven sökväg.

Om användaren exempelvis har navigerat till sökvägen "/customer" kommer denna att matchas mot de tillåtna sökvägarna i routes.json-filen. Utifrån denna matchning kommer routern att skapa en instans av klassen "CustomerController" och sedan åberopa dennes getAll-metod. "CustomerController" ärver egenskaper och metoder av klassen "ParentController".

Metoden getAll kommer i sin tur att skapa en instans av klassen "CustomerModel". "CustomerModel" har flera metoder för att hantera data. Då många av de operationer som "CustomerModel" utför har sin motsvarighet i "CarModel" har den kod som faktiskt kommunicerar med databasen brutits ut och finns i klassen "DataModel" som de övriga "..Model" klasserna ärver från.

"CustomerModel" kommer att i sin tur att förbereda en fråga eller ett statement och sedan åberopa en av de metoder på "DataModel" som direkt kommunicerar med databasen. Genom att dela upp koden på detta vis undviks repetition. "CustomerModel" kommer att returnera det svar denne får från "DataModel" till "CustomerController".

När "CustomerController" kommer att skicka denna information vidare till Twig som använder sig av denna och en template för att skapa den HTML-kod som returneras av servern till användaren.

Databasen

Databasen är en MySQL databas och innehåller fem tabeller. En för data relaterat till kunder, en för bilar, en för uthyrningar och två för bilars möjliga märken och färger.

Tabellen för kunder har personnummer som primärnyckel och tabellen för bilar har registreringsnummer som primärnyckel. Tabellerna för bilmärke och färg innehåller inga nycklar.

Tabellen för uthyrningar har id som primärnyckel vilken är satt att automatiskt öka för varje ny rad. Tabellen innehåller även två främmande nycklar, personnummer och registreringsnummer, vilka är kopplade till primärnycklarna i kund- och biltabellerna.

Exempelkod

Följande sidor innehåller kodexempel ur applikationen. Exemplen är ett urval av vad jag ansett viktigast att visa upp. Övrig kod finns på GitHub och är kommenterad på liknande vis.

Då jag anser att koden är utförligt kommenterad och då jag försökt använda beskrivande namn på variabler, metoder och klasser kommer jag inte att gå in närmare på den här.

```sql
1  DROP database IF EXISTS CarRental;
2  CREATE database CarRental CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
3  USE CarRental;
4
5  CREATE TABLE customers(
6    personnumber BIGINT PRIMARY KEY,
7    name VARCHAR(255) NOT NULL,
8    address VARCHAR(255) NOT NULL,
9    postaladdress VARCHAR(255) NOT NULL,
10   phonenumber VARCHAR(10) NOT NULL
11 ) ENGINE=InnoDb;
12
13 CREATE TABLE cars(
14   registration CHAR(6) PRIMARY KEY,
15   make VARCHAR(100) NOT NULL,
16   color VARCHAR(20) NOT NULL,
17   year INT NOT NULL,
18   price INT NOT NULL
19 ) ENGINE=InnoDb;
20
21 CREATE TABLE rentals(
22   id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
23   registration CHAR(6),
24   personnumber BIGINT,
25   checkouttime DATETIME NOT NULL,
26   checkintime DATETIME,
27   days INT,
28   cost FLOAT,
29   FOREIGN KEY (registration) REFERENCES cars(registration),
30   FOREIGN KEY (personnumber) REFERENCES customers(personnumber)
31 ) ENGINE=InnoDb;
32
33 CREATE TABLE makes(
34   make VARCHAR(100) NOT NULL
35 ) ENGINE=InnoDb;
36
37 CREATE TABLE colors(
38   color VARCHAR(100) NOT NULL
39 ) ENGINE=InnoDb;
40
41 INSERT INTO customers (personnumber, name, address, postaladdress, phonenumber)
   VALUES
42   (199309230465, "Lisa Andersson", "Storgatan 2", "45656 Småstad", "0775343455"),
43   (197002101207, "Mohammad Rezai", "Stackvägen 12", "75646 Uppsala", "0708118825"),
44   (197804282338, "Ali Boraoui", "Hässjevägen 3", "75647 Uppsala", "0722345678"),
45   (198807280030, "Petri Lejerdahl", "August Södermans väg 9", "75645 Uppsala",
   "0725999988"),
46   (194604279796, "Mona Karlsson", "Stenbrohultsvägen 107", "75754 Uppsala",
   "0737882211"),
47   (195704143295, "Mario Garcia", "Valhallavägen 52", "11414 Stockholm",
   "0722994450"),
48   (197405314563, "Lovisa Larsson", "Glimmerstigen 11", "45776 Granitbyn",
   "0711959545"),
49   (196302254344, "Helena Janols", "Drottninggatan 20", "86040 Sundsvall",
   "0703454545"),
50   (197302271452, "Stefan Åhlander", "Lapplandsresan 25 B", "75755 Uppsala",
   "0704979766"),
51   (198007271482, "Maria Erlandsson", "Lapplandsresan 25 B", "75755 Uppsala",
   "0725100580");
52
```

```sql
53  INSERT INTO cars (registration, make, color, year, price) VALUES
54    ("LST364", "Volvo", "Blue", 2019, 700),
55    ("PDG675", "Audi", "Blue", 2016, 500),
56    ("LMS009", "Volvo", "White", 2012, 300),
57    ("KUL456", "Mercedes", "Green", 2018, 700),
58    ("ABC123", "Hyundai", "White", 2015, 350),
59    ("SAS022", "BMW", "Blue", 2019, 800),
60    ("NKM342", "Volvo", "Black", 2015, 400),
61    ("PDF110", "Hyundai", "Yellow", 2017, 450),
62    ("LSW364", "Toyota", "Gray", 2012, 300),
63    ("DOL990", "Toyota", "Red", 2011, 200);
64
65  INSERT INTO makes (make) VALUES
66    ("Peugeot"),
67    ("Suzuki"),
68    ("Fiat"),
69    ("Honda"),
70    ("Ford"),
71    ("Hyundai"),
72    ("Renault"),
73    ("Toyota"),
74    ("Volkswagen"),
75    ("Chrystler"),
76    ("Volvo"),
77    ("Audi"),
78    ("BMW"),
79    ("Mercedes");
80
81    INSERT INTO colors (color) VALUES
82      ("Blue"),
83      ("Red"),
84      ("Green"),
85      ("Yellow"),
86      ("Black"),
87      ("White"),
88      ("Magenta"),
89      ("Gray"),
90      ("Brown");
```

```php
 1 <?php
 2 /**
 3  * The heart of the application. All routes leads to index.php
 4  */
 5 use Main\Core\Config;
 6 use Main\Core\Router;
 7 use Main\Core\Request;
 8 use Main\Utils\DependencyInjector;
 9
10   require_once __DIR__ . "/vendor/autoload.php";
11
12   /**
13    * Once loaded a config object is instantiated. Based on the information
14    * in the config file a database connection is established.
15    */
16   $config = new Config();
17
18   $dbConfig = $config->get('db');
19   $db = new PDO(
20       'mysql: host=' . $dbConfig['host'] . ';dbname=' . $dbConfig['dbname'] .
   ';charset=utf8',
21       $dbConfig['user'],
22       $dbConfig['password']
23   );
24
25   /**
26    * Twig is set up to render views.
27    */
28   $loader = new \Twig\Loader\FilesystemLoader(__DIR__ . "/src/Views");
29   $twig = new \Twig\Environment($loader);
30   $twig->addGlobal('baseUrl', $config->get('baseUrl'));
31
32   /**
33    * The dependency injector is instantiated and populated with the database
   connection,
34    * the config object and Twig.
35    */
36   $di = new DependencyInjector();
37   $di->set('PDO', $db);
38   $di->set('Utils\Config', $config);
39   $di->set('Twig_Environment', $twig);
40
41   /**
42    * The current http request is parsed by the Request object.
43    */
44   $request = new Request();
45
46   /**
47    * The current route is parsed based on the request.
48    */
49   $router = new Router($di);
50
51   /**
52    * A view is rendered based on what controller is called by the route method.
53    */
54   echo $router->route($request, $twig);
55
56 ?>
```

```php
<?php

namespace Main\Domain;

/**
 * Customer object with constructor, getter methods, method to turn
 * object properties into an associative array and methods to
 * set or reset the renting property.
 */
class Customer {
  private $personnumber;
  private $name;
  private $address;
  private $postaladdress;
  private $phonenumber;
  private $renting;

  public function __construct($specs) {
    $this->personnumber = $specs["personnumber"];
    $this->name = $specs["name"];
    $this->address = $specs["address"];
    $this->postaladdress = $specs["postaladdress"];
    $this->phonenumber = $specs["phonenumber"];
    if(isset($specs["renting"])) {
      $this->renting = $specs["renting"];
    } else {
      $this->renting = FALSE;
    }
  }

  public function getPersonNumber() {
    return $this->personnumber;
  }

  public function getName() {
    return $this->name;
  }

  public function getAddress() {
    return $this->address;
  }

  public function getPostalAddress() {
    return $this->postaladdress;
  }

  public function getPhoneNumber() {
    return $this->phonenumber;
  }

  public function getRenting() {
    return $this->renting;
  }

  public function setRenting() {
    return $this->renting = TRUE;
  }

  /**
   * Method to get object properties and create an associative array.
```

```php
61      * For use in ombination with calls to Model-methods.
62      */
63    public function transformToDatabaseAppropriateArray() {
64        $arr["personnumber"] = $this->personnumber;
65        $arr["name"] = $this->name;
66        $arr["address"] = $this->address;
67        $arr["postaladdress"] = $this->postaladdress;
68        $arr["phonenumber"] = $this->phonenumber;
69
70        return $arr;
71    }
72 }
73
```

```php
 1  <?php
 2
 3  namespace Main\Core;
 4
 5  use Main\Controllers\ErrorController;
 6
 7  /**
 8   * Class to manage routes. All different routes go to the index.php file
 9   * where the Request object dissect route information from the url and
10   * POST and GET variables. Based on this information the Router Object
11   * calls the right Controller class and passes on any data.
12   */
13  class Router {
14    private $di;
15    private $routeMap;
16    private static $regexPatters = [
17        'number' => '\d+',
18        'string' => '\w+',
19        'mixed' => '[a-zA-Z0-9]+'
20    ];
21
22    /**
23     * Constructor. Receives a Dependency Injection Object and stores this as
24     * a property to be accessed by all child classes. Also gets all possible routes
25     * from the file routes.json and stores them in a map (an associative array).
26     */
27    public function __construct($di) {
28      $this->di = $di;
29
30      $json = file_get_contents(__DIR__ . '/../../config/routes.json');
31      $this->routeMap = json_decode($json, true);
32  }
33
34    /**
35     * Loops over all possible routes in the stored map and compares them to
36     * the provided URI-path. If there is a match the corresponding controller will
37     * be called and data passed on. If no match is found instantiates a not found
38     * exception and calls the ErrorController.
39     */
40    public function route($request) {
41      $path = $request->getPath();
42
43      foreach ($this->routeMap as $route => $info) {
44        $regexRoute = $this->getRegexRoute($route, $info);
45        if (preg_match("@^/$regexRoute$@", $path)) {
46          return $this->executeController($route, $path, $info, $request);
47        }
48      }
49
50      $errorController = new ErrorController($this->di, $request);
51
52      return $errorController->notFound();
53    }
54
55    /**
56     * Creates a regex pattern from the provided route by pairing optionaly provided
57     * parameters to the route and replacing these with regex-patterns.
58     */
59    private function getRegexRoute($route, $info) {
60      if (isset($info['params'])) {
```

```php
      foreach ($info['params'] as $name => $type) {
        $route = str_replace(':' . $name, self::$regexPatters[$type], $route);
      }
    }

    return $route;
  }

  /**
   * Extract provided parameters by spliting the path and route variables
   * and matching the provided parameters to the corresponding route part in
   * an associative array.
   */
  private function extractParams($route, $path) {
    $params = [];
    $pathParts = explode('/', $path);
    $routeParts = explode('/', $route);

    foreach ($routeParts as $key => $routePart) {
      if (strpos($routePart, ':') === 0) {
        $name = substr($routePart, 1);
        $params[$name] = $pathParts[$key+1];
      }
    }

    return $params;
  }

  /**
   * Call the controller provided by the matching route in the routes map.
   */
  private function executeController($route, $path, $info, $request) {
    $controllerName = '\Main\Controllers\\' . $info['controller'] . 'Controller';
    $controller = new $controllerName($this->di, $request);

    $params = $this->extractParams($route, $path);

    return call_user_func_array([$controller, $info['method']], $params);
  }
}
?>
```

```php
<?php

/**
 * Parent class for all controllers.
 * Sets upp common variables and methods to be inherited by child classes.
 */
namespace Main\Controllers;

use Main\Core\Request;
use Main\Utils\DependencyInjector;

abstract class ParentController {
  protected $request;
  protected $db;
  protected $config;
  protected $view;
  protected $di;

  public function __construct(DependencyInjector $di, Request $request) {
    $this->request = $request;
    $this->di = $di;
    $this->db = $di->get('PDO');
    $this->view = $di->get('Twig_Environment');
    $this->config = $di->get('Utils\Config');
  }

  protected function render($template, $params) {
    return $this->view->render($template, $params);
  }
}
```

```php
<?php

namespace Main\Controllers;

use Main\Models\RentalModel;
use Main\Models\CarModel;
use Main\Models\CustomerModel;
use Main\Core\FilteredMap;

/**
 * Handles almost all rental related functionality as reguested by the provided URI.
 */
class RentalController extends ParentController {

  /**
   * Instantiate both a CarModel object and a CustomerModel object.
   * Use these to get a list of all cars not rented and all customers.
   * Use these lists to populate the 'check out' view with select input
   * elements.
   */
  public function checkOutCar() {
    $carModel = new CarModel($this->db);
    $customerModel = new CustomerModel($this->db);

    try {
      $cars = $carModel->getAllNotRented();
    } catch (\Exception $e) {
      $properties = ['errorMessage' => 'Error creating cars not rented.'];
      return $this->render('error.twig', $properties);
    }

    try {
      $customers = $customerModel->getAll();
    } catch (\Exception $e) {
      $properties = ['errorMessage' => 'Error creating customers.'];
      return $this->render('error.twig', $properties);
    }

    $properties = ["cars" => $cars, "customers" => $customers];
    return $this->render('checkout.twig', $properties);
  }

  /**
   * Instantiate a new RentalModel object. Get person number of the customer
   * renting and the licens number of the car to be rented from the form on the
   * checkout page. Use this information to create a new row in the rentals
   * table in the database. Use the returned id to get the row back from the database
   * showing i.e. checkout time etc. Use this info to populate a view showing the
   * newly created rental.
   */
  public function checkedOutCar() {
    $rentalModel = new RentalModel($this->db);

    $fM =  new FilteredMap($this->request->getForm());
    $personnumber = $fM->getInt("personnumber");
    $registration = $fM->getString("registration");

    try {
      $id = $rentalModel->createRental($personnumber, $registration);
    } catch (\Exception $e) {
```

```php
61          $properties = ['errorMessage' => 'Error creating rental.'];
62          return $this->render('error.twig', $properties);
63        }
64
65        try {
66          $rental = $rentalModel->get($id);
67        } catch (\Exception $e) {
68          $properties = ['errorMessage' => 'Error getting rental by id.'];
69          return $this->render('error.twig', $properties);
70        }
71
72        $properties = ["rental" => $rental];
73        return $this->render('checkedout.twig', $properties);
74      }
75
76      /**
77       * Instantiate a CarModel object. Use a method of the object to get a list of
78       * all cars currently rented. Use the list to populate the checkin view.
79       */
80      public function checkInCar() {
81        $carModel = new CarModel($this->db);
82
83        try {
84          $cars = $carModel->getAllRented();
85        } catch (\Exception $e) {
86          $properties = ['errorMessage' => 'Error getting all rented cars.'];
87          return $this->render('error.twig', $properties);
88        }
89
90        $properties = ["cars" => $cars];
91        return $this->render('checkin.twig', $properties);
92      }
93
94      /**
95       * Instantiate a new RentalModel object. Get the licens plate (registration) of the
96       * car to be checked in from the form element on the checkin view. Pass this
97       * registration to the object method to close the rental. Use the returned id to
98       * get the corresponding row from the database table. Use this information to
99       * populate a view showing the closed rental.
100      */
101     public function checkedInCar() {
102       $rentalModel = new RentalModel($this->db);
103
104       $fM =  new FilteredMap($this->request->getForm());
105       $registration = $fM->getString("registration");
106
107       try {
108         $id = $rentalModel->closeRental($registration);
109       } catch (\Exception $e) {
110         $properties = ['errorMessage' => 'Error closing rental.'];
111         return $this->render('error.twig', $properties);
112       }
113
114       try {
115         $rental = $rentalModel->get($id);
116       } catch (\Exception $e) {
117         $properties = ['errorMessage' => 'Error getting rental by id.'];
118         return $this->render('error.twig', $properties);
119       }
120
```

```
121        $properties = ["rental" => $rental];
122        return $this->render('checkedin.twig', $properties);
123    }
124
125    /**
126     * Get and show all rental history.
127     */
128    public function getHistory() {
129        $rentalModel = new RentalModel($this->db);
130
131        try {
132            $rentals = $rentalModel->getAll();
133        } catch (\Exception $e) {
134            $properties = ['errorMessage' => 'Error getting rental history.'];
135            return $this->render('error.twig', $properties);
136        }
137
138        $properties = ["rentals" => $rentals];
139        return $this->render('history.twig', $properties);
140    }
141 }
```

```php
1  <?php
2
3  namespace Main\Controllers;
4
5  use Main\Domain\Customer;
6  use Main\Models\CustomerModel;
7  use Main\Models\RentalModel;
8  use Main\Core\FilteredMap;
9
10 /**
11  * Handles almost all customer related functionality as reguested by the provided
    URI.
12  */
13 class CustomerController extends ParentController {
14
15   /**
16    * Instantiate the CustomerModel object and call it's getAll function.
17    * If there is an error calling the CustomerModel method populate the
18    * 'properies' variable with an error message and render the
19    * error view using Twig.
20    * Call a method to get all customers who are currently renting a
21    * car. Loop over the results and combine these with the previous
22    * list of all customers to get the complete customer objects to render on screen.
23    * Pass the list of customers to the render method of Twig.
24    */
25   public function getAll() {
26     $customerModel = new CustomerModel($this->db);
27
28     try {
29       $customers = $customerModel->getAll();
30     } catch (\Exception $e) {
31       $properties = ['errorMessage' => 'Error getting customers from Controller.'];
32       return $this->render('error.twig', $properties);
33     }
34
35     $properties = ['customers' => $customers];
36     //die(var_dump($properties));
37
38     return $this->render('customers.twig', $properties);
39   }
40
41   /**
42    * Render the add customer view.
43    */
44   public function add() {
45     $properties = [];
46     return $this->render('addcustomer.twig', $properties);
47   }
48
49   /**
50    * Get form data from the 'add customer' view form element and call the
51    * CustomerModel with this information to create a new row in the database
52    * representing this new customer.Render view to show the customer that was added.
53    */
54   public function addedCustomer() {
55     $customerModel = new CustomerModel($this->db);
56     $newCustomer = $this->getCustomerFromForm();
57
58     try {
59       $addedCustomer = $customerModel->create($newCustomer);
```

```php
60        } catch (\Exception $e) {
61           $properties = ['errorMessage' => 'Error creating customer.'];
62           return $this->render('error.twig', $properties);
63        }
64
65        $properties = ["customer" => $addedCustomer];
66        return $this->render('addedcustomer.twig', $properties);
67     }
68
69     /**
70      * Get the person number of the customer to be edited from the hidden form element.
71      * This is not passed in the URI because of it's sensitive nature!!
72      *
73      * Call the get method in the CustomerModel to get all customer information
74      * from the database. Use this info to populate the view.
75      */
76     public function editCustomer() {
77        $customerModel = new CustomerModel($this->db);
78        $fM =  new FilteredMap($this->request->getForm());
79        $personnumber = $fM->getInt("personnumber");
80
81        try {
82           $customer = $customerModel->get($personnumber);
83        } catch (\Exception $e) {
84           $properties = ['errorMessage' => 'Error getting customer.'];
85           return $this->render('error.twig', $properties);
86        }
87
88        $properties = ["customer" => $customer];
89        return $this->render('editcustomer.twig', $properties);
90     }
91
92     /**
93      * Get form data from the 'edit customer' view form element and call the
   CustomerModel
94      * with this information to create a new row in the database representing this new
95      * customer. Render view to show the customer that was edited.
96      */
97     public function editedCustomer() {
98        $customerModel = new CustomerModel($this->db);
99        $newCustomer = $this->getCustomerFromForm();
100
101        try {
102           $editedCustomer = $customerModel->edit($newCustomer);
103        } catch (\Exception $e) {
104           $properties = ['errorMessage' => 'Error editing customer.'];
105           return $this->render('error.twig', $properties);
106        }
107
108        $properties = ["customer" => $editedCustomer];
109        return $this->render('editedcustomer.twig', $properties);
110     }
111
112     /**
113      * Get the person number of the customer to be deleted from the hidden form
   element.
114      * This is not passed in the URI because of it's sensitive nature!!
115      *
116      * Call the CustomerModel to get all information about the customer to be deleted.
```

```php
117       * Also call the RentalModel to remove all occurances of the customer from the
    rentals table.
118       * Finally call the CustomerModel to delete the customer from the database. Use the
    previously
119       * gotten information about the customer to populate a view showing what customer
    has been
120       * deleted. Using this information offers the possibility to undo the delete action
121       * by calling the 'added customer' function by navigation that route from the
    'deleted view'.
122       * Unfortunately the removal of all occurances from the rentals table can not be
123       * rolled back the same way.
124       *
125       */
126     public function deleteCustomer() {
127         $customerModel = new CustomerModel($this->db);
128         $rentalModel = new RentalModel($this->db);
129         $fM =  new FilteredMap($this->request->getForm());
130         $personnumber = $fM->getInt("personnumber");
131
132         try {
133             $customer = $customerModel->get($personnumber);
134         } catch (\Exception $e) {
135             $properties = ['errorMessage' => 'Customer not found.'];
136             return $this->render('error.twig', $properties);
137         }
138
139         try {
140             $rentalModel->removeCustomer($personnumber);
141         } catch (\Exception $e) {
142             $properties = ['errorMessage' => 'Error removing customer from rental
    history.'];
143             return $this->render('error.twig', $properties);
144         }
145
146         try {
147             $customerModel->delete($personnumber);
148         } catch (\Exception $e) {
149             $properties = ['errorMessage' => 'Error deleteing customer.'];
150             return $this->render('error.twig', $properties);
151         }
152
153         $properties = ["customer" => $customer];
154         return $this->render('deletedcustomer.twig', $properties);
155     }
156
157     /**
158      * Helper function to get form data. Also instantiates a new customer object
159      * based on this data.
160      */
161     private function getCustomerFromForm() {
162         $fM =  new FilteredMap($this->request->getForm());
163         $customer["personnumber"] = $fM->getInt("personnumber");
164         $customer["name"] = $fM->getString("name");
165         $customer["address"] = $fM->getString("address");
166         $customer["postaladdress"] = $fM->getString("postaladdress");
167         $customer["phonenumber"] = $fM->getString("phonenumber");
168
169         return new Customer($customer);
170     }
171 }
```

172

```php
<?php

namespace Main\Models;

use Main\Exceptions\NotFoundException;
use Main\Exceptions\DbException;

/**
 * Medel handling transactions with the database. Provides generic
 * methods used by child classes to indirectly storing and
 * manipulating data. This provides encapsulation and abstraction.
 */
class DataModel {
  protected $db;

  /**
   * Constructor connecting the database object to the class.
   */
  public function __construct($db) {
    $this->db = $db;
  }

  /**
   * Execute specified query.
   *
   * @param { $query = statement to be executed on the database.}
   *
   * @return  { The raw results of the query.}
   */
  public function executeQuery($query, $specs = [], $amendBy = "") {
    $sth = $this->db->prepare($query . $amendBy);
    if (!$sth->execute($specs)) {
      throw new DbException($sth->errorInfo()[2]);
    }

    return $sth->fetchAll();
  }

  /**
   * Get one row from table.
   *
   * @param { $specs = :array containing the associative key => values to be
  interpolated
   *                   on the query.
   *          $specs["table] = what table to get results from,
   *          $specs["column] = what column to serch by,
   *          $specs["value"] = what value to match for.
   *
   *          $amendBy = optional string to amend the standard query.}
   *
   * @return  { The raw results of the query.}
   */
    public function getGeneric($specs, $amendBy = "") {
    $query = "SELECT * FROM " . $specs["table"] . " WHERE " .
             $specs["column"] . " = :value" . $amendBy;

    $sth = $this->db->prepare($query . $amendBy);

    if (!$sth->execute(["value" => $specs["value"]])) {
      throw new DbException($sth->errorInfo()[2]);
```

```php
 60       }
 61
 62       $results = $sth->fetchAll();
 63
 64       if (empty($results)) {
 65         throw new NotFoundException($specs["value"] . " not found in " .
     $specs["column"]);
 66       }
 67
 68       return $results[0];
 69     }
 70
 71   /**
 72    * Get all rows from table.
 73    *
 74    * @param { $specs["table] = what table to get results from,
 75    *         $specs["order] = order results by.}
 76    *
 77    * @return  { The raw results of the query.}
 78    */
 79   public function getAllGeneric($specs) {
 80     $query = "SELECT * FROM " . $specs["table"] . " ORDER BY " . $specs["order"];
 81     $sth = $this->db->prepare($query);
 82
 83     if (!$sth->execute()) {
 84       throw new DbException($sth->errorInfo()[2]);
 85     }
 86
 87     return $sth->fetchAll();
 88   }
 89
 90   /**
 91    * Execute insert or update query.
 92    *
 93    * @param { $query = statement to be executed on the database,
 94    *         $specs = :array containing the associative key => values  to be
 95    *                  interpolated on the query. Number of keys nust match
 96    *                  exactly to the query.}
 97    *
 98    * @return  { The last insert id.}
 99    */
100   public function insertOrUpdateGeneric($query, $specs) {
101     $sth = $this->db->prepare($query);
102
103     if (!$sth->execute($specs)) {
104       throw new DbException($sth->errorInfo()[2]);
105     }
106
107     return $this->db->lastInsertId();
108   }
109
110   /**
111    * Delete on or more rows from table.
112    *
113    * @param { $specs = :array containing the associative key => values to be
     interpolated
114    *                    on the query.
115    *         $specs["table"] = what table to delete from,
116    *         $specs["column"] = what column to match by,
117    *         $specs["value"] = what value to match for.}
```

```php
118        */
119        public function deleteGeneric($specs) {
120        $query = "DELETE FROM " . $specs["table"] . " WHERE " .
121        $specs["column"] . " = :value";
122        $sth = $this->db->prepare($query);
123
124        if (!$sth->execute(["value" => $specs["value"]])) {
125            throw new DbException($sth->errorInfo()[2]);
126        }
127      }
128
129      /**
130       * Function to remove all occurances of a specified value from a
131       * specified table and column.
132       *
133       * @param { $specs = :array containing the associative key => values to be
    interpolated
134       *                       on the query.
135       *          $specs["table] = what table to modify,
136       *          $specs["column] = what column to remove value from,
137       *          $specs["value"] = what value to remove.}
138       */
139      public function removeOccurance($specs) {
140        $query = "UPDATE " . $specs["table"] . " SET " . $specs["column"] .
141        " = NULL WHERE "  . $specs["column"] . " = :value";
142        $sth = $this->db->prepare($query);
143
144        if (!$sth->execute(["value" => $specs["value"]])) {
145            throw new DbException($sth->errorInfo()[2]);
146        }
147      }
148
149 }
```

```php
1  <?php
2
3  namespace Main\Models;
4
5  use Main\Domain\Rental;
6  use Main\Domain\Car;
7
8  /**
9   * Class for handling primarily transactions with the database on table rentals.
10  * Some complicatded transactions affecting the cars tables are also handled
11  * here because they are part of larger operations involving multiple tables.
12  */
13 class RentalModel extends DataModel {
14
15   /**
16    * Get a specific rental transaction by id.
17    *
18    * @param { $id = id for the rental transaction to get.}
19    *
20    * @return  { New Rental object.}
21    */
22   public function get($id) {
23     $result = parent::getGeneric([
24       "table" => "rentals",
25       "column" => "id",
26       "value" => $id]);
27
28     return new Rental($result);
29   }
30
31   /**
32    * Get all rentals by calling parent class getAll method.
33    *
34    * @return  { Array of new Rental objects.}
35    */
36   public function getAll() {
37     $results = parent::getAllGeneric([
38       "table" => "rentals",
39       "order" => "checkouttime"
40     ]);
41
42     foreach($results as $result) {
43       $rentals[] = new Rental($result);
44     }
45
46     return $rentals;
47   }
48
49   /**
50    * Create and store a new rental transaction.
51    *
52    * @param { $personnumber = person number of the customer who is
53    *                          renting.
54    *         $registration = licens plate of the rented car.}
55    */
56   public function createRental($personnumber, $registration) {
57     // Create a new rental transaction in the rentals table.
58     $query = <<<SQL
59 INSERT INTO rentals (registration, personnumber, checkouttime, checkintime, days,
   cost)
```

```php
60  VALUES (:registration, :personnumber, CONVERT_TZ(NOW(),  @@session.time_zone,
    "Europe/Stockholm"), null, null, null)
61  SQL;
62
63      $specs = ["personnumber" => $personnumber,
64              "registration" => $registration];
65      $id = $this->executeInsertOrUpdate($query, $specs);
66
67      return $id;
68    }
69
70    /**
71     * Close a currently open rental transaction.
72     *
73     * @param { $registration = licens plate of the rented car.}
74     */
75    public function closeRental($registration) {
76      //  Get the rental from the rentals table based on registration and set
    variables.
77      $specs = ["table" => "rentals",
78              "column" => "registration",
79              "value" => $registration];
80      $amendBy =" AND checkintime IS NULL";
81
82      $rental = new Rental($this->getGeneric($specs, $amendBy));
83
84      $id = $rental->getID();
85
86      //  Get info about the car from the cars table to get the rental price.
87      $specs = ["table" => "cars",
88              "column" => "registration",
89              "value" => $registration];
90
91      $car = new Car(parent::getGeneric($specs));
92
93      $price = $car->getPrice();
94
95      //  Start transaction.
96      $this->db->beginTransaction();
97
98      //  Set the checkin time in rentals table.
99      $query = <<<SQL
100 UPDATE rentals
101 SET checkintime = CONVERT_TZ(NOW(),  @@session.time_zone, "Europe/Stockholm")
102 WHERE id = :id
103 SQL;
104
105      $specs = ["id" => $id];
106      $this->executeInsertOrUpdate($query, $specs);
107
108      // Calculate the number of days the car has been rented by using mysql-diff
    function.
109      $query = <<<SQL
110 SELECT TIMESTAMPDIFF(SECOND,
111   (SELECT checkouttime FROM rentals WHERE id = :id),
112   (SELECT checkintime FROM rentals WHERE id = :id))
113 SQL;
114
115      $specs = ["id" => $id];
116      try {
```

```php
117        $diff = parent::executeQuery($query, $specs)[0][0];
118      } catch (\Exception $e) {
119        $this->db->rollBack();
120        throw $e;
121      }
122
123      if($diff <= 86400) {
124        $days = 1;
125      } else {
126        $days = intval($diff / 86400) + 1;
127      }
128
129      $cost = strval($days * $price);
130
131        // Set number of days rented and total cost in the rentals table.
132      $query = <<<SQL
133 UPDATE rentals
134 SET days = :days, cost = :cost
135 WHERE id = :id
136 SQL;
137
138      $specs = ["id" => $id,
139              "days" => $days,
140              "cost" => $cost];
141      $this->executeInsertOrUpdate($query, $specs);
142
143      // Commit all transactions and return id from rentals table.
144      $this->db->commit();
145
146      return $id;
147    }
148
149    /**
150     * Function to remove a specific customers person number from the rentals table
151     * by calling parent class generic removeOccurance method.
152     */
153    public function removeCustomer($personnumber) {
154      parent::removeOccurance([
155        "table" => "rentals",
156        "column" => "personnumber",
157        "value" => $personnumber]);
158    }
159
160    /**
161     * Function to remove a specific cars registration number from the rentals table
162     * by calling parent class generic removeOccurance method.
163     */
164    public function removeCar($registration) {
165      parent::removeOccurance([
166        "table" => "rentals",
167        "column" => "registration",
168        "value" => $registration]);
169    }
170
171    /**
172     * Helper function to avoid repetition. Use within transaction to be able to
173     * rollback any changes if something throws an error.
174     *
175     * @param { $query = the query passed on from the calling function.
176     *          $specs = the specs passed on from the calling function.}
```

```php
177      */
178    public function executeInsertOrUpdate($query, $specs) {
179      try {
180        $id = parent::insertOrUpdateGeneric($query, $specs);
181      } catch (\Exception $e) {
182        $this->db->rollBack();
183        throw $e;
184      }
185      return $id;
186    }
187 }
```

```php
1  <?php
2
3  namespace Main\Models;
4
5  use Main\Domain\Customer;
6
7  /**
8   * Model for hadling customer data providing a simple interface
9   * for the CustomerController. Extends and uses the DataModel
10  * for communication with the database.
11  */
12 class CustomerModel extends DataModel {
13
14   /**
15    * Get ustomers by calling parent class get method.
16    *
17    * @param { $personnumber = person number for the customer to get.}
18    *
19    * @return  { New Customer object.}
20    */
21   public function get($personnumber) {
22     $result = parent::getGeneric([
23       "table" => "customers",
24       "column" => "personnumber",
25       "value" => $personnumber]);
26
27     return new Customer($result);
28   }
29
30   /**
31    * Get all customers by calling parent class getAll method.
32    *
33    * @return  { Array of new Customer objects.}
34    */
35   public function getAll() {
36     $query = <<<SQL
37       SELECT personnumber, name, address, postaladdress, phonenumber,
38       IF ((SELECT checkouttime FROM rentals
39       WHERE customers.personnumber = rentals.personnumber
40       AND checkouttime IS NOT NULL AND checkintime IS NULL)
41       IS NOT NULL, TRUE, FALSE) AS renting FROM customers
42 SQL;
43
44     $specs = [
45       "table" => "customers",
46       "order" => "name"
47     ];
48
49     $results = parent::executeQuery($query, $specs);
50
51     foreach($results as $result) {
52       $customers[] = new Customer($result);
53     }
54
55     return $customers;
56   }
57
58   /**
59    * Insert (create) new customer in the customers table by calling the parent class
60    * create method.
```

```php
 61      *
 62      * @param { $customer = customer object to create new row from.}
 63      *
 64      * @return  { Customer object created from the inserted data.}
 65      */
 66     public function create($customer) {
 67         $query = <<<SQL
 68 INSERT INTO customers(personnumber, name, address, postaladdress, phonenumber)
 69 VALUES(:personnumber, :name, :address, :postaladdress, :phonenumber)
 70 SQL;
 71
 72         $specs = $customer->transformToDatabaseAppropriateArray();
 73
 74         parent::insertOrUpdateGeneric($query, $specs);
 75
 76         return $this->get($customer->getPersonNumber());
 77     }
 78
 79     /**
 80      * Update (modify) customer in the customers table by calling the parent class
 81      * edit method.
 82      *
 83      * @param { $customer = customer object to base the update on.}
 84      *
 85      * @return  { Customer object created from the updated data.}
 86      */
 87     public function edit($customer) {
 88         $query = <<<SQL
 89 UPDATE customers
 90 SET name=:name, address=:address, postaladdress=:postaladdress,
    phonenumber=:phonenumber
 91 WHERE personnumber=:personnumber
 92 SQL;
 93
 94         $specs = $customer->transformToDatabaseAppropriateArray();
 95
 96         parent::insertOrUpdateGeneric($query, $specs);
 97
 98         return $this->get($customer->getPersonNumber());
 99     }
100
101     /**
102      * Delete customer by calling parent class delete method.
103      *
104      * @param { $personnumber = person number for the customer to delete.}
105      */
106     public function delete($personnumber) {
107         parent::deleteGeneric([
108             "table" => "customers",
109             "column" => "personnumber",
110             "value" => $personnumber]);
111     }
112 }
113
```

```twig
 1  <!DOCTYPE html>
 2  <html lang="en">
 3    <head>
 4      <meta charset="UTF-8" />
 5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 6      <meta http-equiv="X-UA-Compatible" content="ie=edge" />
 7      <link rel="stylesheet" type="text/css" href="{{ baseUrl }}/styles/theme.css">
 8      <title>{% block title %}{% endblock %}</title>  </head>
 9    <body>
10      <header class="background-dark">
11        <h1><a href="./">Sunshine Car Rental</a></h1>
12      </header>
13
14      <nav>
15        <form method="post" action="{{ baseUrl }}/customers">
16          <input type="submit" value="Customers">
17        </form>
18        <form method="post" action="{{ baseUrl }}/cars">
19          <input type="submit" value="Cars">
20        </form>
21        <form method="post" action="{{ baseUrl }}/checkout">
22          <input type="submit" value="Check out car">
23        </form>
24        <form method="post" action="{{ baseUrl }}/checkin">
25          <input type="submit" value="Check in car">
26        </form>
27        <form method="post" action="{{ baseUrl }}/history">
28          <input type="submit" value="History">
29        </form>
30      </nav>
31
32      <main>
33        {% block contentTitle %}{% endblock %}
34        {% block content %}{% endblock %}
35      </main>
36
37      <script src="{{ baseUrl }}/js/main.js"></script>
38    </body>
39  </html>
40
```

```twig
{% extends 'layout.twig' %}


{% block content %}
  <table>
    <thead>
      <th>Person Number</th>
      <th>Name</th>
      <th>Address</th>
      <th>Postal Address</th>
      <th>Phonenumber</th>
      <th></th>
      <th></th>
    </thead>

  {% block innerContent %}
  {% endblock %}
{% endblock %}
```

```twig
{% extends 'customers-layout.twig' %}

{% block title %}
    Customers
{% endblock %}

{% block contentTitle %}
  <h2>Customers</h2>
{% endblock %}

{% block innerContent %}

{% for customer in customers %}
    <tr>
      <td>{{ customer.personnnumber }}</td>
      <td>{{ customer.name }}</td>
      <td>{{ customer.address }}</td>
      <td>{{ customer.postaladdress }}</td>
      <td>{{ customer.phonenumber }}</td>
      <td>
        <form method="post" action="{{ baseUrl }}/editcustomer">
          <input style="display: none" name="personnumber" type="text" value="{{ customer.personnumber }}" >
          <input type="submit" value="Edit" {% if customer.renting == true %}disabled{% endif %}>
        </form>
      </td>
      <td>
        <form class="delete" method="post" action="{{ baseUrl }}/deletecustomer">
          <input style="display: none" name="personnumber" type="text" value="{{ customer.personnumber }}" >
          <input type="submit" value="Delete" {% if customer.renting == true %}disabled{% endif %}>
        </form>
      </td>
    </tr>
{% endfor %}

</table>
  <form method="post" action="{{ baseUrl }}/addcustomer">
    <input type="submit" value="Add customer">
  </form>
{% endblock %}
```

```javascript
 1  /**
 2   * Object containing information on various selectors for elements on which to add
 3   * event listners and event handlers.
 4   *
 5   * selector = selector for elements to attach event listner to
 6   * event = what kind of event should trigger handler
 7   * handler = function to handle event
 8   */
 9  const elementIdSet = [
10    {
11      selector: '#validate',
12      event: 'submit',
13      handler: validate
14    },
15    {
16      selector: '#selectMake',
17      event: 'change',
18      handler: removeChild
19    },
20    {
21      selector: '#selectColor',
22      event: 'change',
23      handler: removeChild
24    },
25    {
26      selector: '.delete',
27      event: 'submit',
28      handler: checkBeforeDelete
29    }
30  ];
31
32  /**
33   * Object containing methods to validate various input data.
34   * Every input field that should be validated has a data-validate attribute contaning
35   * the name of the corresponding method. Forms that should be validated have the
    validate event handler attached.
36   */
37  const validateMethods = {
38    personnumber(elt) {
39      let returnMessage = '';
40
41      let num = (elt.value = elt.value.replace(/[-\s]/g, ''));
42      if (!/^\d{12}$/.test(num)) {
43        return `Person number should be 12 figures and only numbers.\n`;
44      }
45
46      let time = new Date(
47        ('' + num)
48          .match(/^(\d{4})(\d{2})(\d{2})/)
49          .slice(1)
50          .join('-')
51      ).getTime();
52
53      if (isNaN(time)) {
54        returnMessage += `Person number should start with a valid date in the format
    YYYYMMDD.\n`;
55      }
56
57      let check = [...num.slice(2)]
58        .map((n, index) => (index % 2 ? 1 : 2) * n)
```

```javascript
 59       .join('')
 60       .split('')
 61       .reduce((acc, cur) => acc + +cur, 0);
 62
 63     if (check % 10 !== 0) {
 64       returnMessage += `The control number for person nummer is incorrect.\n`;
 65     }
 66     return returnMessage;
 67   },
 68   phonenumber(elt) {
 69     let returnMessage = '';
 70
 71     let num = (elt.value = elt.value.replace(/[-\s]/g, ''));
 72     if (!/^0\d{9}$/.test(num)) {
 73       returnMessage += `Phonenumber should be 10 figures, start with 0 and only
 contain numbers.\n`;
 74     }
 75     return returnMessage;
 76   },
 77   registration(elt) {
 78     let returnMessage = '';
 79     elt.value = elt.value.replace(/[a-z]/gi, l => l.toUpperCase());
 80
 81     let num = elt.value.replace(/[-\s]/g, '');
 82     if (!/^[a-hj-pr-uw-z]{3}\d{2}[a-hj-npr-uw-z0-9]{1}$/i.test(num)) {
 83       returnMessage += `The registration doesn\'t comply with Swedish standards.\n`;
 84     }
 85     return returnMessage;
 86   },
 87   year(elt) {
 88     let returnMessage = '';
 89
 90     let num = (elt.value = elt.value.replace(/[-\s]/g, ''));
 91     if (!/^\d{4}$/.test(num) || num < 1900 || num > 2020) {
 92       returnMessage += `The registration year should be a whole number between 1900
 and 2020.\n`;
 93     }
 94     return returnMessage;
 95   },
 96   price(elt) {
 97     let returnMessage = '';
 98
 99     let num = (elt.value = elt.value.replace(',', '.'));
100     if (!/^[\d.]+$/.test(num)) {
101       returnMessage += `The the price should only contain numbers and an optional
 separator.\n`;
102     }
103
104     num = Number.parseFloat(num).toFixed(2);
105     if (isNaN(num) || num < 0) {
106       returnMessage += `The price should be a positive number.\n`;
107     }
108     return returnMessage;
109   }
110 };
111
112 /**
113  * Function that loops over all input fields that should be validated and calls the
114  * corresponding methods and compiles an error message in case of errors.
115  *
```

```javascript
116   * @param { element that has the triggered event handler}
117   */
118  function validate(e) {
119    const list = document.querySelectorAll('[data-validate]');
120    let message = '';
121
122    list.forEach(elt => {
123      message += validateMethods[elt.dataset.validate](elt);
124    });
125
126    if (message) {
127      e.preventDefault();
128      alert('Your form contains errors!\n' + message);
129      return false;
130    }
131  }
132
133  /**
134   * Removes any chiled element that has the remove class.
135   */
136  function removeChild() {
137    this.querySelector('.remove') && this.querySelector('.remove').remove();
138  }
139
140  /**
141   * Check to make sure the user really wants to delete the item.
142   *
143   * @param {element of the triggered event} elt
144   */
145  function checkBeforeDelete(elt) {
146    console.log('checking to delete');
147    let check = confirm('Are you shure you want to delete this?');
148    if (!check) {
149      elt.preventDefault();
150      return false;
151    }
152    return true;
153  }
154
155  /**
156   * Loop over list of event handlers and attach events to present elements.
157   */
158  elementIdSet.forEach(item => {
159    document
160      .querySelectorAll(item.selector)
161      .forEach(elt => elt.addEventListener(item.event, item.handler));
162  });
163
```