Semesterprojekt Software Engineering 1

Meine Variante des digitalen Parkhauses wurde basierend auf dem Architekturpattern des Model-View-Controller umgesetzt. Es existiert ein zentraler Controller (ParkhausController.java – *Autor Max Kettenhofen*) der ein Modell und zwei Views verwaltet. Das Modell, welches ebenso ein Singleton darstellt, ist die Klasse ParkhausSystem.java (*gemischte Autorenschaft*). In dem Modell werden alle anfallenden Daten verarbeitet oder an eine Statistikklasse (Statistics.java – *gemischte Autorenschaft*) weitergeleitet. Der Controller ist zudem für die Ein- und Ausparkfunktionalität zuständig. Die Einparkfunktionalität wird von den Klassen ParkDefault.java und ParkBackward.java (*Autor Patrick Nussbaum, Anpassungen Max Kettenhofen*) bestimmt, welche die Reihenfolge der Parkplatzbelegung an das System weitergeben. Hierbei werden Parkplätze der Reihe nach von vorne oder hinten gefüllt. Sollte ein Parkplatz freiwerden, so wird dieser wieder zuerst gefüllt.

Die Klasse ParkhausServlet.java (*gemischte Autorenschaft)* ist für die Übermittlung der Daten zwischen den Klassen und der aktiven Session zuständig. Je nach gewählter Seite stehen dem Nutzer des Parkhauses verschiedene Methoden zur Verfügung. Hierbei wird unterschieden zwischen der „Customer“-Seite und der „Admin“-Seite, wobei letztere mehr Funktionalität bezüglich Statistiken bietet.

Der Customer kann nur Fahrzeuge ein- und ausparken und sich die bisher angefallenen Parkkosten anzeigen lassen. Dies wird realisiert durch die Klasse ViewCustomer.java (*Autor Max Kettenhofen*). Hierbei wird bei der Aktivierung der Funktion eine Notification durch das System an die Observer/Views ViewCustomer und ViewAdmin (*Autor Max Kettenhofen*) weitergeleitet. In der Klasse ViewCustomer wird nun die aktuelle Systemzeit entnommen und mit der zwischengespeicherten Einfahrtszeit des zuletzt eingefahrenen Fahrzeugs verglichen. Hieraus wird der Preis berechnet, wobei der Typ des Fahrzeugs beachtet wird.

Es existieren fünf verschiedene Fahrzeugtypen. Intern werden diese jedoch alle als „Car“ repräsentiert. Die Klasse Car.java (*Autor Patrick Nussbaum, Anpassungen Max Kettenhofen*) ist hierbei eine Implementierung des Multiton-Patterns, eine weiterführende Variante des Singleton. Fahrzeuge besitzen einen Key, der den Typ des Fahrzeugs repräsentiert. Fahrzeuge werden beim Einfahren in das Parkhaus nur instanziiert, falls es das erste Fahrzeug seiner Art ist. Ansonsten wird basierend auf dem Key die entsprechende Instanz aus der statischen ConcurrentMap entnommen. Somit wird nicht bei jedem Parkvorgang ein neues Fahrzeug erstellt, welches nach dem Ausparken entsorgt oder gespeichert werden muss. Alle relevanten Daten können trotzdem zur Berechnung der geforderten Bilanzen und Werte entnommen und gespeichert werden. Die geschieht beim Ein- und Ausparken des Fahrzeugs. Die übergebenen Infos wie Einfahrtszeit, Fahrzeugtyp, Preis etc. werden vom System verwaltet.

View Nr. 2 ist ViewAdmin.java (*Autor Max Kettenhofen*). Diese Ansicht kann über den Admin-Button auf der Customer-Seite aufgerufen werden und wird über eine zweite .jsp Seite gesteuert. Als Admin hat man Zugriff auf eine Aufsummierung aller bisherigen Gewinne, einer Auflistung der durchschnittlichen Parkkosten und -zeiten, sowie eine Vielzahl an Graphen. Drei der Graphen werden von der Klasse Graphs.java (*Autor Patrick Nussbaum, Anpassungen Max Kettenhofen*) verwaltet. Diese verwenden JSON-Strukturen, um die erforderlichen Daten mittels Plotly anzuzeigen. Die Klasse wird vom Servlet mit den aktuellen Daten aus dem System aufgerufen. Je nach übergebenem Typstring des Graphen wird eine andere Methode zur Abarbeitung der Daten aufgerufen, die mittels Stream() und ArrayList() die Daten verarbeitet und zum Output zusammenstellt.

Der finale Button der Adminseite ist ebenfalls ein Plotly-Graph, dieser wird jedoch nicht durch die Graphen-Klasse erzeugt, sondern gehört zur Klasse ViewAdmin. Hierbei werden die Einnahmen für den Tag und die Woche aggregiert und als Barchart dargestellt. Hierzu wird ähnlich wie vorher die aktuelle Systemzeit bei Bezahlung eines Tickets, also beim Ausparkvorgang mit CurrentTimeMillis() entnommen und wird nun mit der Erstellungszeit der Session verglichen, welche in der init()-Methode des Servlets gespeichert wurde. Ist die Differenz zwischen den Zeiten geringer als ein Tag, so wird der Preis zu Tages- und Wocheneinnahmen gezählt. Ist die Differenz jedoch höher, so wird der Tag zurückgesetzt und die Hochzählung beginnt für die Tageseinnahmen erneut. Ein Tagescounter wird hierbei ebenfalls inkrementiert. Sollte dieser 8 erreichen wird ebenfalls die Woche zurückgesetzt.

Getestet wurden die Ein- und Ausparkfunktionalität und die Statistikerhebung in der Klasse ParkhausSystemTest.java (*Autor Patrick Nussbaum, Anpassungen Max Kettenhofen*).