Case Study OSM: RedCars

OSM-M team
studiejaar 2019/2020; semester 1; v2.00

Inleiding

Autoverhuurbedrijf RentIt wil gaan starten met het aanbieden van deelauto's. Deze service gaan ze RedCars noemen. In plaats van auto's centraal op te moeten halen bij RentIt zelf, wil RentIt de auto's gaan plaatsen waar de vraag is. Om RedCars goed te laten functioneren wil RentIt een applicatie laten ontwikkelen.

RentIt vraagt aan ICA-studenten om een analyse en ontwerp te maken voor RedCars. Hieronder wordt de functionaliteit beschreven.

Er zijn een paar randvoorwaarden aan deze opdracht verbonden: je moet in groepjes van twee studenten een objectgeoriënteerde analyse en ontwerp in UML maken. Doe dit op een wijze waarop de iedere stap een logisch gevolg is op de voorgaande stappen. Je ontwerp werk je ook uit in C++ met minimaal alle in UML opgenomen onderdelen.

Registreren

Personen kunnen zich als klant registreren. Hierbij moeten Naam, Adres en Woonplaats opgegeven worden. Daarnaast wordt ook gevraagd naar een emailadres en bankrekeningnummer. Voor dit bankrekeningnummer wordt ook goedkeuring verleend voor automatische afschrijving, omdat RedCars na gebruik automatisch de betaling hiervan int. Aangemelde personen krijgen een pasje thuisgestuurd. Met het emailadres als gebruikersnaam en het pasnummer als wachtwoord kunnen ze nu inloggen in het systeem.

Beheer van gegevens

Door medewerkers van RentIt kunnen de klantgegevens beheerd worden. Alle gegevens van klanten kunnen worden aangepast. Daarnaast kunnen klanten inactief gemaakt worden. Dit kan nodig zijn als klanten b.v. slecht betalen of auto's met nieuwe gebreken achterlaten.

Naast klanteninfo kan ook autoinformatie worden beheerd. Van auto's worden kenteken, type en standplaats vastgelegd. Op de meeste plaatsen staat een kleine personenauto, maar op enkele plaatsen staat een wat grotere stationwagen met trekhaak.

Gebruik van de auto

Klanten kunnen op een website kijken waar de auto's geparkeerd staan per stad en of die auto beschikbaar is. Dit moet erg eenvoudig en snel kunnen, ook als er veel klanten tegelijkertijd boeken. De klant kan de auto ook op de website reserveren. Hierbij geeft hij de begin- en eindtijd op van de reservering.

Klanten kunnen voor de periode dat ze gereserveerd hebben de auto meenemen. Bij iedere parkeerlocatie staat een paal waar kan worden in- en uitgecheckt. (Op populaire locaties staan meerdere auto's. Voor ieder van deze auto's kan je in- en uitchecken bij dezelfde paal.). Iedere auto is uitgerust met een module waar ondermeer GPS-tracking aanwezig is. Dankzij deze module kunnen bestuurders de auto openen en afsluiten en de auto starten met de pas. Natuurlijk kan een auto pas worden geopend nadat is ingecheckt. Om te zorgen dat klanten niet vergeten om uit te checken nadat ze de auto hebben afgesloten op de RedCars parkeerplaats, geven auto en paal lichtsignalen totdat is uitgecheckt.

RedCars wil dat auto's zo goed mogelijk door meerdere klanten gebruikt worden. Om die reden wil RedCars dat klanten zo precies mogelijk aangeven wanneer zij de auto gebruiken. Zo ziet RedCars graag dat klanten die een auto voor bijvoorbeeld vier dagen nodig hebben deze voor vier dagen reserveren. Klanten kunnen dan zelf het meer aantrekkelijke weektarief kiezen in plaats van viermaal het dagtarief te betalen. Om dit gedrag verder te stimuleren is het niet toegestaan om de auto te parkeren op de RedCars parkeerplaats zonder uit te checken. Met andere woorden, zolang je de auto in gebruik hebt mag je deze niet weer parkeren op de eigen parkeerplaats, of op een andere RedCars parkeerplaats.

Betaling

Een belangrijk onderdeel van de applicatie is het berekenen van de kosten. Hiervoor zijn een aantal zaken van belang. Een klant kan uit meerdere vormen van abonnementen kiezen en daarnaast uit meerdere types auto's. Berekening van kosten vindt plaats op basis van gegevens van de reservering. Bij overschrijding van de gereserveerde periode wordt het huurbedrag als boete berekend en worden de extra uren vervolgens nog apart per uur verrekend.

Hieronder vind je een initieel idee voor abonnementen en kosten per type auto. Zowel types van auto's als abonnementen moeten erg eenvoudig uitbreidbaar zijn en de berekening daarop aangepast.

Abonnement	**Gratis**		**Betaald**	
Type auto	Personen	Station	Personen	Station
per uur	€ 6,00	€ 7,50	€ 4,00	€ 5,50
per dag	€ 50	€ 60	€ 40	€ 50
per weekend	€ 70	€ 90	€ 60	€ 80
per week	€ 150	€ 180	€ 140	€ 170

km vrij	–	–	100	100

Daarnaast wordt er ook per kilometer afgerekend:

Abonnement	**Gratis**		**Betaald**	
Type auto	Personen	Station	Personen	Station
per km	€ 0,30	€ 0,35	€ 0,25	€ 0,30

Om verder misbruik te waarborgen, is het niet mogelijk een auto te reserveren en in te checken zolang er een betalingsachterstand is. Een betalingsachterstand ontstaat wanneer een automatische afschrijving mislukt. De automatische afschrijvingen worden overigens ondergebracht bij een externe partij. Het interface met deze externe partij moet door ons worden ontworpen.

Nu we het toch over misbruik hebben, het mag niet mogelijk zijn om op hetzelfde tijdstip meerdere malen ingecheckt te zijn.

Opdracht

Je wordt gevraagd om de analyse, het ontwerp en een deel van de realisatie uit te voeren in de vorm van een SRS, SDD en C++ broncode.

Om de hoeveelheid werk te beheersen hoef je alleen het geraamte van de C++ code te realiseren. Dit houdt in dat alle componenten (exe en dll), klassen, functies etc zoals beschreven in het ontwerp aanwezig zijn. User interfaces (GUI), databases en externe systemen moeten als mock object worden opgenomen. De communicatie tussen componenten moet gedemonstreerd kunnen worden met behulp van console berichten. Minimale logica van user input tot actuatoren is daarvoor noodzakelijk.

Te nemen stappen (volgorde naar eigen inzicht):

1. Stel een use case model op dat bestaat uit een use case diagram en de use case beschrijvingen in brief format
2. Stel een overzicht op van overige eisen ingedeeld volgens FURPS. Zorg er in ieder geval voor dat je minimaal de niet-functionele hints uit de tekst allemaal “smart” hebt opgenomen.
3. Beschrijf de use cases die horen bij het het reserveren, ophalen van de auto, het terugbrengen van de auto en de betaling van de huur in fully dressed format.
4. Maak een domeinmodel op basis van de informatie uit deze opdracht. Vul deze zonodig aan zodat in ieder geval alles aanwezig is dat relevant is voor alle brief use cases en uitgewerkte fully-dressed use cases.
5. Ontwerp een deploymentdiagram. Later kan dit diagram nog aangepast worden met nieuwe inzichten.

6. Ontwerp een componentdiagram. Geef in eerste instantie naar beste vermogen goede namen aan de interfaces, pas de interfaces later aan op basis van nieuwe inzichten.
7. Werk de fully-dressed use case(s) uit met behulp van system sequence diagrams, (component) sequence diagrams en class diagrams.
8. Maak naar eigen inzicht gebruik van activity en state diagrams.
9. Realiseer het ontwerp in C++ op basis van de eerder genoemde richtlijnen.

Tips

- Mochten er onduidelijkheden zijn over deze casus, neem dan contact op met de opdrachtgever. De docent zal deze rol op zich nemen.
- Vraag tussendoor feedback over keuzes die je hebt gemaakt met betrekking tot de diagrammen en modellen die je opstelt.
- Zorg voor coherentie tussen de verschillende onderdelen van je analyse, ontwerp en realisatie. Daar waar je bewust afwijkt van coherentie tussen de diagrammen is de afwijking en de reden ervoor toegelicht.

Beoordeling

Je werk wordt op de volgende onderdelen beoordeeld (verslag is het SRS en SDD):

1. Leesbare verslaglegging, dit betekent in ieder geval:
    - Je verslag bestaat uit tekst met diagrammen ter ondersteuning. Je verslag is dus géén verzameling diagrammen.
    - Diagrammen in je verslag zijn leesbaar in een geprinte versie van je verslag.
    - Er is een toelichting bij de diagrammen (denk aan gemaakte keuzen, overwogen opties, afwijkingen tussen een stap en de volgende, bedenkingen bij je gevonden oplossing)
2. Syntactisch correcte diagrammen in je ontwerp.
3. Coherentie tussen de verschillende onderdelen van je analyse en ontwerp.
4. Implementatie in overeenstemming met je ontwerp.
5. Kwaliteit van je analyse en ontwerp.
6. Verantwoorde toepassing van OO principes en design patterns in je ontwerp.
7. Mate van volledigheid en zorg.

De eerste twee criteria zijn knock-out criteria.