Ontwerpen en bouwen van software

Een werkend voorbeeld

Frans Spijkerman
Opleiding Informatica
Academie voor Deeltijd
Avans Hogeschool

Versie 0.65

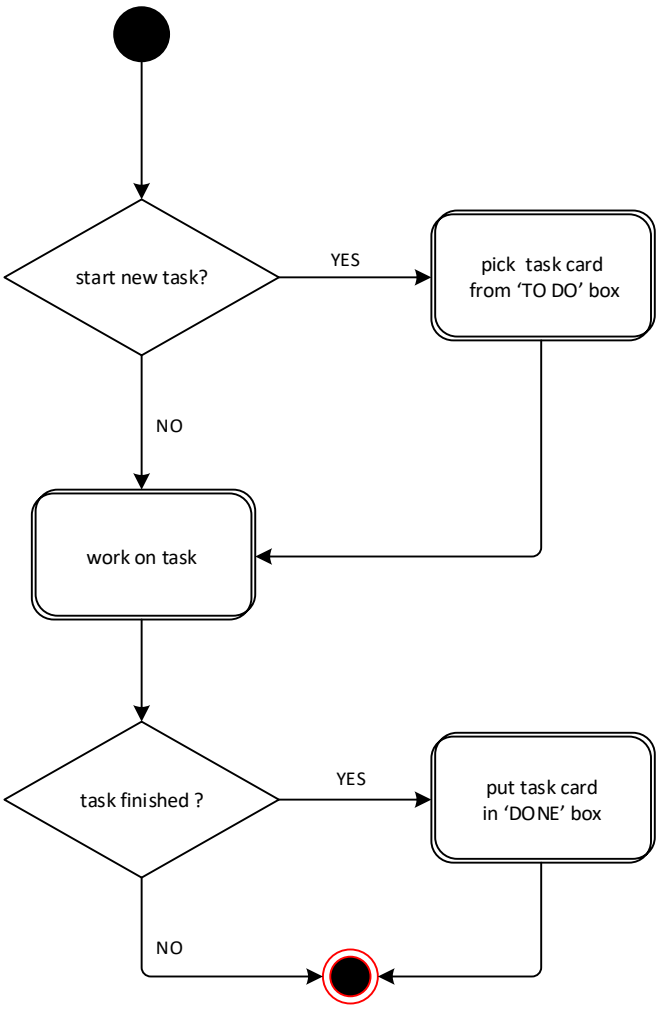
13 december 2022

Bedrijfsproces

De huidige situatie

Medewerkers in ons bedrijf kunnen zelf kiezen welke klussen ze uitvoeren. Hiervoor hebben we twee kaartenbakken met klussen; een met de naam TO DO en een met de naam DONE. Een medewerker kiest een klus uit de bak TO DO en houdt die bij zich. Als de klus geklaard is, moet de bijbehorende kaart in de bak DONE gezet worden. Elke klus hoort bij een project; een project kan meerder klussen bevatten.

Alleen een manager kan nieuwe kaarten in de bak TO DO zetten. We maken in de beschrijving van de bedrijfsprocessen onderscheid tussen *managers* en *klussers (medewerkers)*. De activiteitendiagrammen hieronder beschrijven de administratie rond het werken aan een klus voor een klusser.

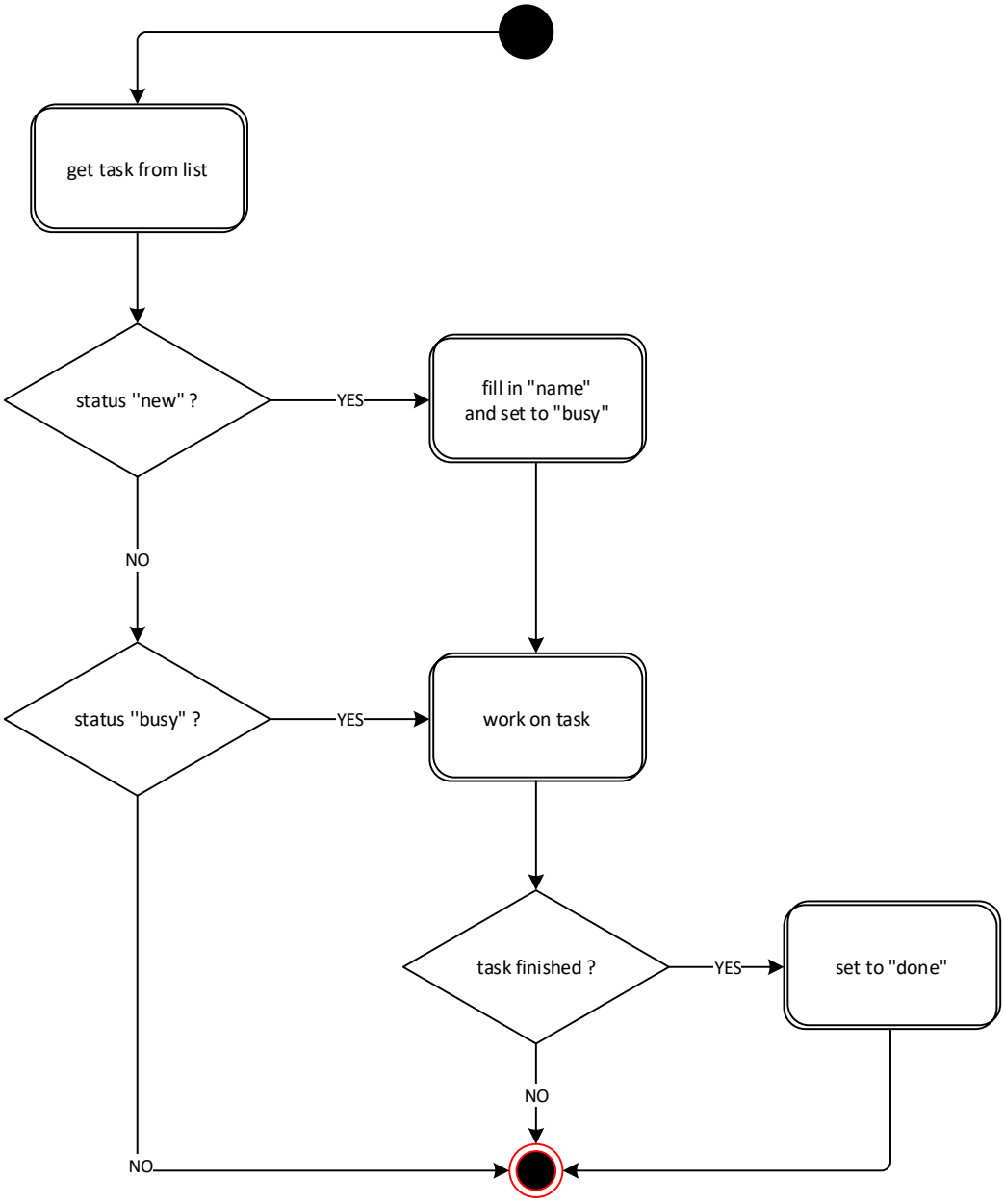


*Figuur 1. Activiteitendiagram: de administratie van werk aan een klus door een klusser in de **huidige situatie***

De gewenste situatie

Managers willen meer inzicht in de status van de verschillende klussen. Met het kaartenbaksysteem is niet duidelijk wie er met welke klus bezig is en de bakken zijn niet van afstand benaderbaar. Het

idee is om een online lijstje met klussen te maken, ingedeeld per project. Een manager kan projecten en klussen toevoegen en bekijken. Het hierboven beschreven administratieve proces voor een medewerker veranderd wel wat: een fysieke kluskaart kan in één van de bakken zitten, of in het bezit van een klusser zijn. In het nieuwe systeem krijgt elke klus in plaats hiervan een status: TO DO, BUSY of DONE.



*Figuur 2. Activiteitendiagram: de administratie van werk aan een klus door een klusser in de **nieuwe situatie***

Requirements

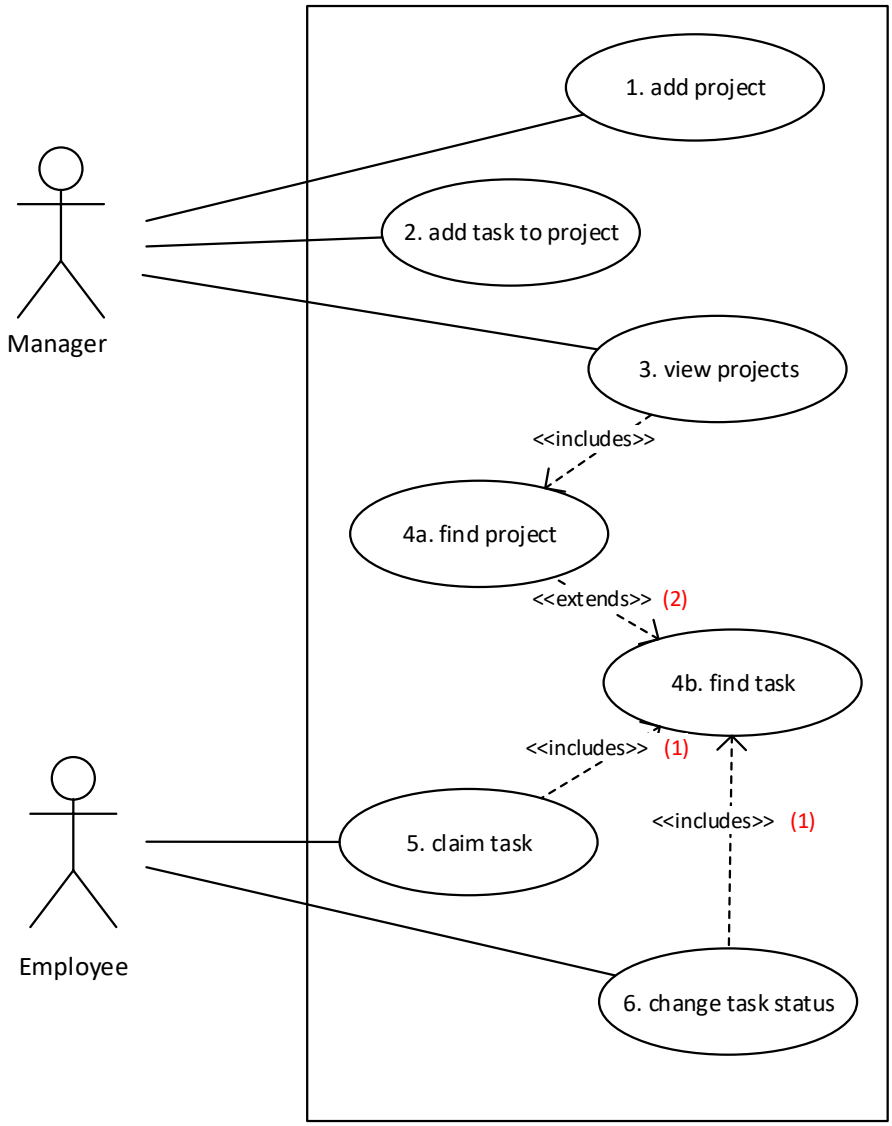
Uit de bovenstaande beschrijving hebben we in eerste instantie de volgende requirements afgeleid.

1. Een manager kan projecten toevoegen
2. Een manager kan klussen aan een project toevoegen
3. Een manager kan een overzicht van projecten en klussen krijgen
4. Manager en klusser kunnen door de lijst van klussen bladeren
5. Een klusser kan intekenen voor een klus
6. Een klusser kan de status van een klus veranderen

De nummers die we aan de requirements geven, houden we zoveel mogelijk aan in de rest van het ontwerp. Dit is handig voor de reqirements traceability. Je vindt de nummers rechtstreeks terug in de nummering van use-cases en bijbehorende testcases, of je gebruikt een traceability matrix.

Use-cases

De gevonden requirements resulteren in een aantal use-cases. Zoals hierboven beschreven kan een medewerker met het nieuwe systeem klussen zoeken en de status van een klus veranderen. Alleen managers kunnen projecten en klussen toevoegen.



Figuur 3. Use Case Diagram van het klussensysteem

In een Use Case Diagram geef je aan wat de gebruikers met het systeem kunnen doen. Niet alle activiteiten uit het Activity Diagram horen in het systeem. Het daadwerkelijk uitvoeren van een klus valt buiten het administratieve systeem, en moet dus niet als use-casediagram of -beschrijving worden opgenomen.

(1) Om een klus te claimen of de status van een klus te wijzigen, moet je eerst de taak zoeken in het systeem, vandaar het gebruik van includes.

(2) Je kunt rechtstreeks in de lijst met klussen zoeken, maar je kunt ook eerst een project zoeken en daarbinnen naar een klus zoeken, vandaar extends (let op de richting van de pijlen)

Use-casebeschrijvingen

We werken één van de use-cases uit in een use-casebeschrijving.

Use-case	**6. Status van klus bijwerken**
Samenvatting	Een werknemer zet de status van een klus op BUSY bij het begin van een klus, of op DONE aan het einde van een klus.
Voorwaarde	Er staan klussen in het systeem
Stappen	1. Werknemer zoekt een klus inde lijst (use-case 5)
	2. Werknemer klikt op 'edit' bij de klus
	3. Systeem geeft details van klus weer in een formulier
	4. Werknemer verandert status van klus van TO DO naar BUSY, of van BUSY naar DONE
	5. Werknemer klikt op 'save'
Resultaat	De status van de klus is aangepast.
Uitzonderingen	Geen

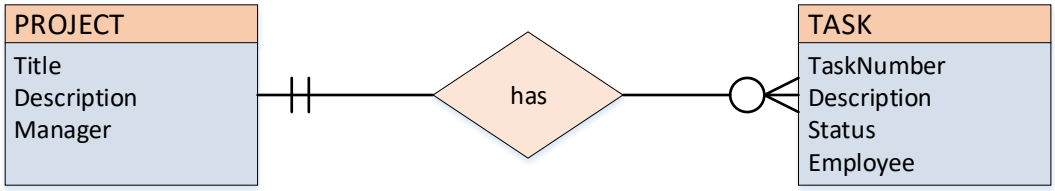
Use-casebeschrijving voor de use-case 'Status van klus bijwerken'

Data

In dit project ontwerpen we voor de gegevens eerst de database.

Entiteiten

We onderscheiden twee entiteiten in het systeem: de projecten (projects) en de klussen (tasks). We hebben besloten dat een project een titel en een beschrijving heeft. Een manager is eigenaar van een project. Een project kan nul of meer klussen bevatten. Elke klus heeft een volgnummer, een beschrijving en de status TO DO, BUSY of DONE. De naam van de klusser die aan de klus begint, wordt natuurlijk ook bewaard.



Figuur 4. Entity Relation Diagram voor het klussensysteem.

Tabellen

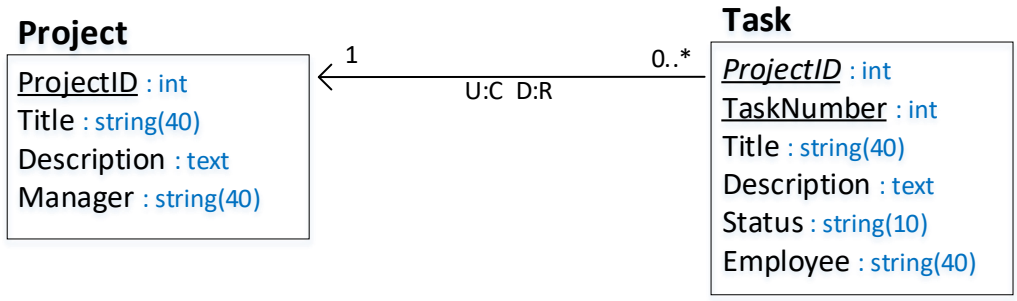
De entiteiten worden uiteindelijk in databasetabellen opgeslagen. Voordat we de nodige tabellen technisch gaan beschrijven met DML, denken we eerst na over hoe we de relaties gaan vastleggen. Hiervoor zetten we het ERD om naar een relationeel model.

Belangrijk hierbij dat we aangeven hoe we de relaties uit het ERD willen oplossen. De één-op-veel-relatie lossen we op met de vreemde sleutel (foreign key) ProjectID in de tabel Task, die verwijst naar een bestaand project. Met onderstreping markeren we de primaire sleutel in een tabel. De vreemde sleutel is met cursief aangegeven.

We bepalen ook de referentiële integriteit. We hebben gekozen voor U:C, zodat een ProjectID ongestraft gewijzigd zou kunnen worden in de projecttabel (zal waarschijnlijk niet gebeuren). D:R is

belangrijker. Het zorgt ervoor dat er geen projecten verwijderd kunnen worden als er klussen aan verbonden zijn.

Het kan geen kwaad om nu ook al over gegevenstypen na te denken. Dit kun je ook uitstellen tot de volgende stap.



Figuur 5. Relationeel model voor de project- en klusgegevens

De database

Dit relationeel model kan bijna maar op één manier uitgelegd worden. De enige toevoeging is de keuze voor automatische nummering van projecten (auto increment). Het klusnummer (tasknumber) kan vrij worden ingevuld, zodat de volgorde veranderd kan worden, maar moet wel uniek zijn binnen een project.

Project heeft dus een enkelvoudige primaire sleutel **ProjectID**. We kiezen voor AUTO_INCREMENT, omdat de sleutel verder geen betekenis heeft. **Task** heeft een samengestelde primaire sleutel **ProjectId** + **TaskNumber**, waarbij **ProjectId** ook de vreemde sleutel is. **TaskNumber** is niet AUTO_INCREMENT, zodat het vrij te kiezen is.

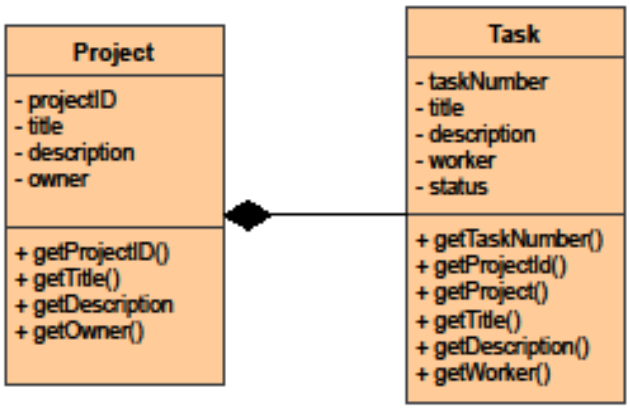
```
CREATE TABLE `Project` (
      `ProjectID` INT PRIMARY KEY AUTO_INCREMENT,
      `Title` VARCHAR(40),
      `Description` TEXT,
      `Manager` VARCHAR(40)
);

CREATE TABLE `Task` (
      `ProjectId` INT,
      `TaskNumber` INT,
      `Title` VARCHAR(40),
      `Description` TEXT,
      `Employee` VARCHAR(40),
      PRIMARY KEY (`ProjectID`, `TaskNumber`),
      FOREIGN KEY  (`ProjectID`) REFERENCES `Project`(`ProjectID`)
);
```

Data Definition Language voor het maken van de tabellen in MySql of MariaDB.

Programmeren

Omdat we een objectgeoriënteerd systeem willen bouwen, gaan we de database niet rechtstreeks benenaderen vanuit het systeem, maar gaan we data-abstractieobjecten gebruiken, DAO's. Hiervoor gaan we eerst klassen definiëren voor de gedefinieerde entiteiten: de domein- of modelklassen.

We zorgen dat alle data-attributen in de klassen verborgen (private) zijn.



Figuur 6. Klassendiagram van de modelklassen

Volledig klassendiagram:
https://raw.githubusercontent.com/spijkerbak/project-manager-1/master/doc/Class_Diagram.pdf

In PHP kan dit resulteren in zoiets:

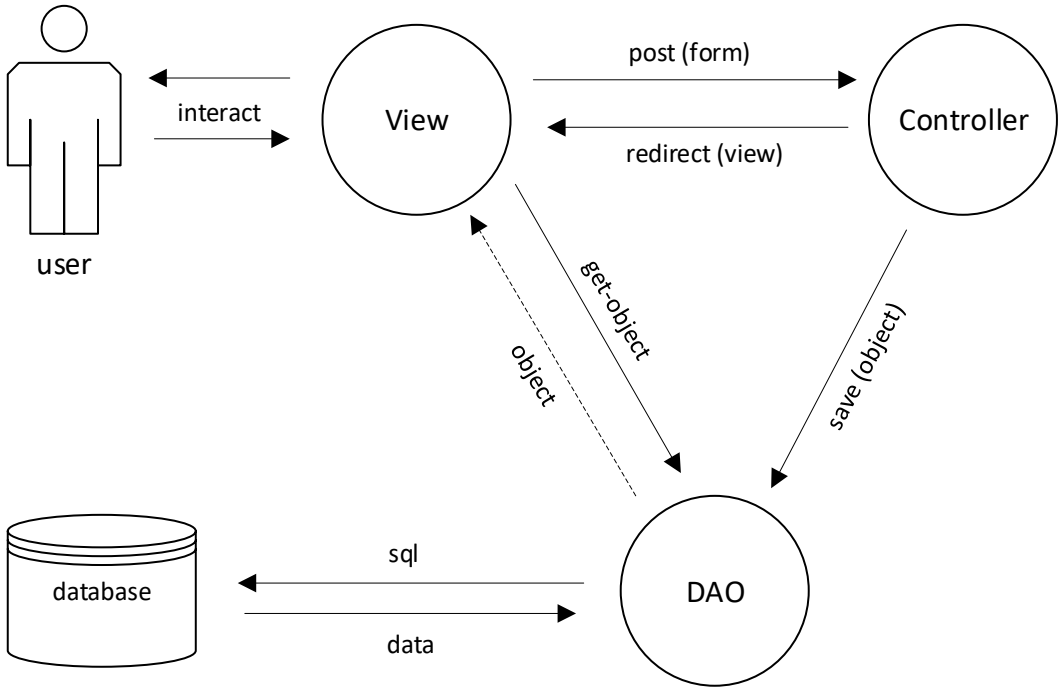
```
class Project {
   private $projectId;
   private $title;
   private $description;
   private $manager;
   private $tasks = []; // project knows its tasks
}

class Task {
   private $project; // task knows its project
   private $number;
   private $title;
   private $description;
   private $status;
   private $employee;
}
```

Maar tijdens het programmeren kunnen we tot andere inzichten komen.

Zie de documentatie en sources op https://github.com/spijkerbak/project-manager-1.

Architectuur

We gaan uit van een variant van MVC, model-view-controller. Een gebruiker gebruikt views in een browser. Deze view kan gegevens ophalen via data-access-objecten. Vanuit een view kan een andere view geactiveerd worden, bijvoorbeeld door een klik op een menukeuze of een wijzigknop. Als er wijzigingen gemaakt moeten worden, zal er een view met een formulier verschijnen. Een formulier wordt naar een controller gepost. De controller vertaalt de formulierdata naar acties voor één of meer data-access-objecten. Een DAO verzorgt de verbinding met de database. Na de database-afhandeling zal de controller via een http-redirect zorgen voor een update van de view.



Figuur 7 Werking van onze MVC-architectuur

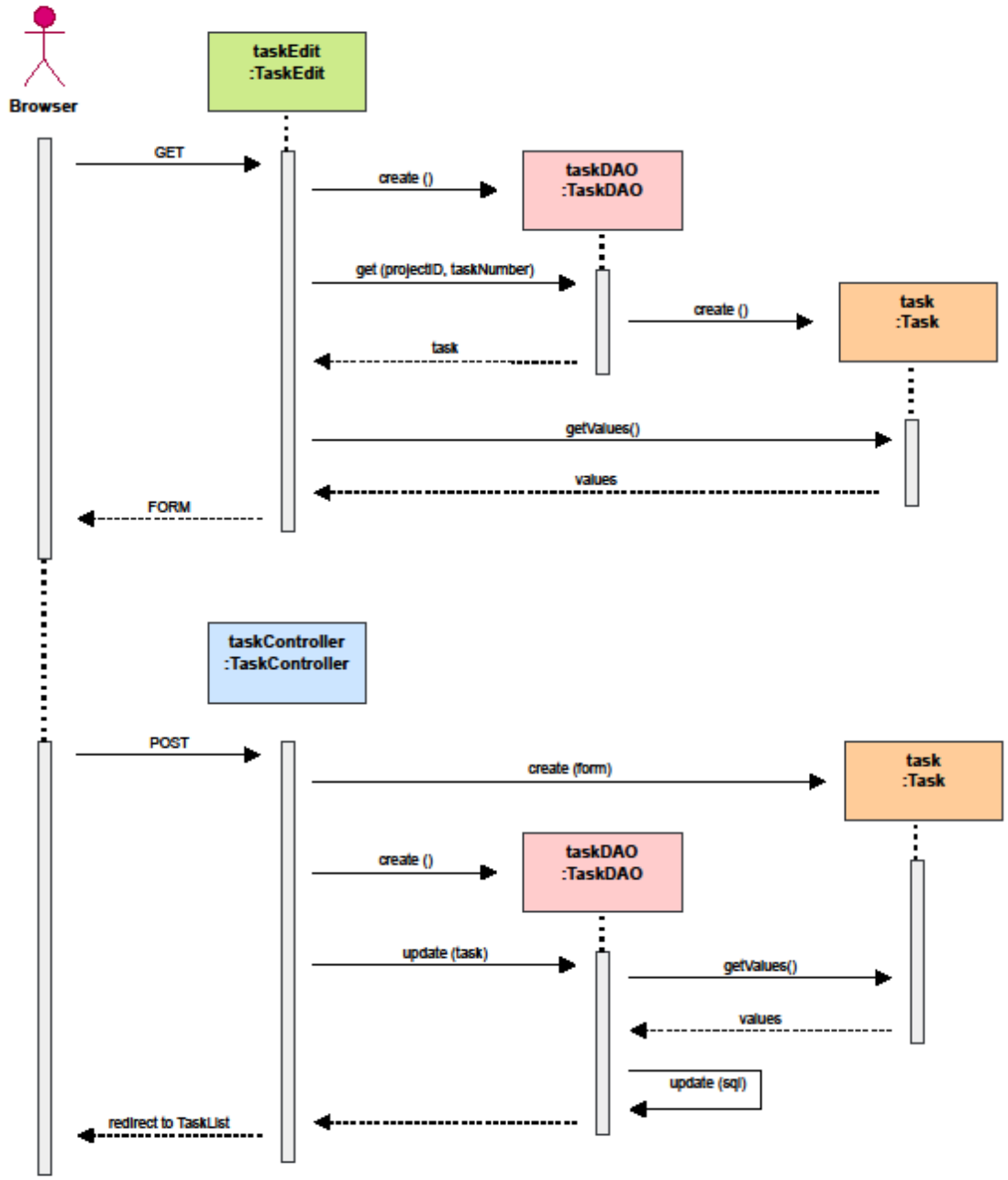
Eigenaardigheden in deze manier van werken:

- Een data–accessobject (dao) wordt gemaakt als het nodig is.
- Modelobjecten worden door de dao's gemaakt.
- Modelobjecten zijn in ons model **immutable**. Ze worden alleen gebruikt om gegevens te bewaren en door te geven. Modelobjecten hebben hier geen setters en verder weinig anders dan getters. Het aanmaken en vullen van objecten gebeurt op drie manieren:
    1. Via de pdo–methode fetchObject, gebruikt om objecten uit de database te halen
    2. Leeg, met new()
    3. Vanuit een formulier, via new ($_POST).

De inhoud van records wordt voorlopig alleen via formulieren gewijzigd.

Sequence diagram

Doorgaans maak je per use case een sequence diagram. Zo'n diagram heeft meestal wat uitleg nodig. Hieronder is use case "6. Status van klus bijwerken" uitgewerkt. Het diagram bestaat uit twee delen.



Figuur 8 Sequentiediagram: Status van taak wijzigen. Leesbaar exemplaar: Sequence_Diagram_6.pdf op Github

Bovenste helft: de gebruiker haal de taakgegevens op en krijgt deze in de vorm van een formulier gepresenteerd. Onderste helft: de gebruiker stuurt het ingevulde formulier naar de server.