Dokumentation

# Syfte och Överväganden

Syftet med denna webbapplikation är att kunna se över och hantera boter som har

förvärvats av användaren. Användaren av applikationen skall kunna hämta, acceptera

och betala, eller bestrida en given bot via applikationen.

Jag la vikt på att lära mig de verktyg, ramverk, tillvägagångssätt,

och dokumentation som skulle krävas under ett faktiskt arbete, då med att jag

såg att det var mer användbart för mig ifall jag blir anställd,

och mer användbart för er i evalueringsprocessen.

Detta dock gjorde att jag inte hade så mycket tid att implementera en slutförd

lösning. Min tidigare erfarenhet var att bara hitta första tillgängliga

lösningen, men såg inte detta som hållbart för denna uppgift givet kontexten.

# Design

## Databas

Jag började med att installera Lavarel Homestead, och fixade så att sidan

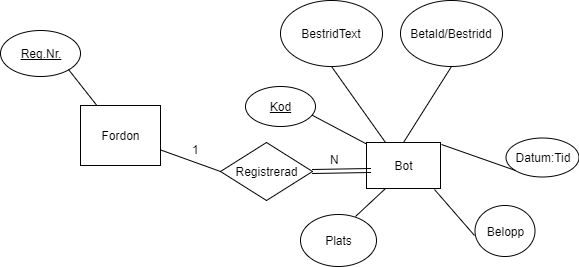
ovningsuppgift.app blev tillgänglig. Jag sedan bytte Lavarel version, och

läste på lite hur den användes. Sedan ändrade jag version på Lavarel, och

startade upp denna applikation. Jag sedan såg över hur jag skulle designa

databasen enligt ER-modellen, och design-mönstret på applikationen.

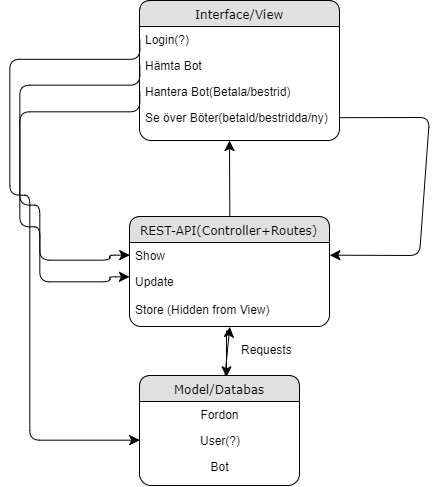
Såhär såg ER-modellen ut av Databasen Botdb:



*Ingen User krävs då botkoden fungerar som lösen. Ett fordon kan ha flera böter på sig men måste åtminstone ha en. Reg.nr ligger i relationsmodellen både i fordon och i bot som foreign key. Users och passwords\_reset tabellerna kommer med i lavarel/databasen men används inte i applikationen, så är därför inte med i datamodellen.*

Jag korrigerade sedan i env. Variabeln så att källkoden kunde kommunicera med databasen.

## High-level/Konceptuell design



Uppdatering (2017-09-18):

User och login behövs ej. Får se om en create-funktion behövs eller ifall detta får matas in helt via SQL. Store-funktionen behövs ju tekniskt sett inte heller, men kan vara nödvändig för att generera random primary key till böter (se Botcontroller). En edit i REST-APIN behövs också nog.

# Metod

## Rest-API

Här börjar jag med att använda artisan migrations kommandot, exempelvis för en bot:

php artisan make:migration create\_bot\_table

Detta gör jag inuti den virtuella maskinen (VirtualBox, via Homestead/Vagrant) och modiferar php-filerna. Detta ska facilitera ändring av andra arbetskamrater. Lavarel Schema är också inbyggt vilket gör att andra databaser skall kunna interagera med den.

Sedan matar jag in tabellerna i databasen med kommandot:

php artisan migrate

Sedan la jag till två controllers. De styr logiken bakom de abstraktionslager för de ORM APIn skall utnyttja för att kommunicera/skicka requests till databasen.

php artisan make:controller Fordon

php artisan make:controller Bot

modifierade sedan dessa så de var restful/resource baserade, och ändrade vilka funktioner som behövde en view eller inte. Fixade också routes så att urls matchar med controller actions i web.php filen:

Route::resource('bot', 'Botcontroller');

Route::resource('fordon', 'Fordoncontroller',

['except' => ['edit', 'update']]);

Skapade också Eloquent ORM modeller via migrate:

php artisan make:model Fordon

php artisan make:model Bot

och modiferade deras innehåll. Det är via dessa resurs-instansers metoder vi lägger till och modifierar data i databasen.

Sen modfierade jag

## Angular Front-End

## CSS(?)

# Section 2