**Bikes Project**

* 1. **Descriere**

Aplicatia Bikes este un sistem online de management al programărilor și stocurilor unui operatii de service de biciclete.

Fiecare client va consulta calendarul service-ului, iar apoi va completa un formular cu data și ora dorită, plus detalii despre problemă, putând adăuga inclusiv fișiere media: imagini și filme.

Administratorul afacerii va putea respinge programarea adăugând un mesaj explicativ: "Ne pare rău, dar nu avem în stoc piesele necesare pentru reparație. Reveniți în S săptămâni" sau o va putea aproba, oferind și un preț estimativ.

De asemenea, aplicația îi va permite acestuia să țină evidența stocurilor existente, cât și a comenzilor date către furnizori. Acesta va putea importa atat date în format CSV și JSON, cât și exporta în oricare din formatele CSV, JSON, PDF.

Aplicatia va fi capabila sa creeze o lista cu reparatii, pretul va fi calculat automat. Dupa realizarea reparatiei, stocul va fi updatat automat.

De asemenea, fiecare element din stoc va avea un "stoc minim", iar atunci cand se ajunge la acesta, se va creea automat o comanda noua pentru acel element. In acest mod stocul nu va ramane niciodata gol si nu se va ajunge la situatia de a refuza o comanda din cauza stocului indisponibil.

Administratorul va putea vizualiza si castigurile totale pe o anumita luna pentru a vizualiza productivitatea afacerii.

Detalii de implementare:

-clientul va consulta zilele si orele disponibile service-ului, va completa formularul cu ora si data dorita, nume, mail, imagine/video si descriere. Programarea va fi preluata si stocata in baza de date( in tabela Programari) ( data si orele disponibile vor fi preluate tot din baza de date)

-administratorul poate vedea programarile in pagina de programari (acestea sunt afisate din baza de date )

-administratorul poate tine evidenta stocurilor din pagina stocuri, el poate adauga/ elimina produse din baza de date (tabela stocuri)

-administratorul poate tine evidenta comenzilor in pagina comenzi, el poate da comenzi noi pentru produsele necesare service-ului

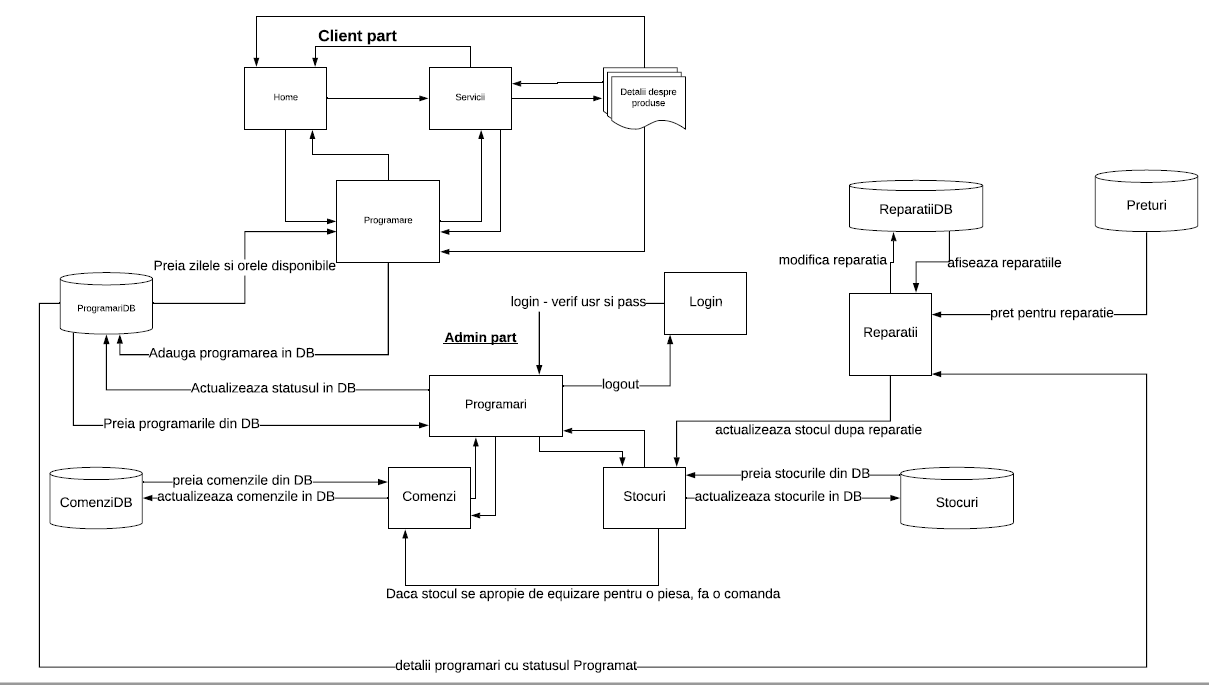
- am adaugat o tabela noua, numita "preturi", care va avea stocat id, reparatie, piesele necesare reparatiei si pretul reparatiei (Ex. 1 - pana - cauciuc - 30 lei )

- am adaugat o tabela noua, numita "reparatii", care va avea stocat id, id\_programare, id\_pret.

- am adaugat o pagina noua "reparatii" care va afisa reparatiile si se vor putea modifica reparatiile( vor fi afisate , reparatiile necesare(pana, schimbat furca), pret( 80 lei), data cand fi facuta reparatia si butoane de adauga elimina)

- reparatiile vor fi generate pe baza descrierii clientului, administratorul va putea adauga si alte reparatii necesare daca este cazul.. sau elimina

- atunci cand administratorul accepta o programare, automat se va creea o "reparatie" in baza de date



**2. Baza de date relationala**

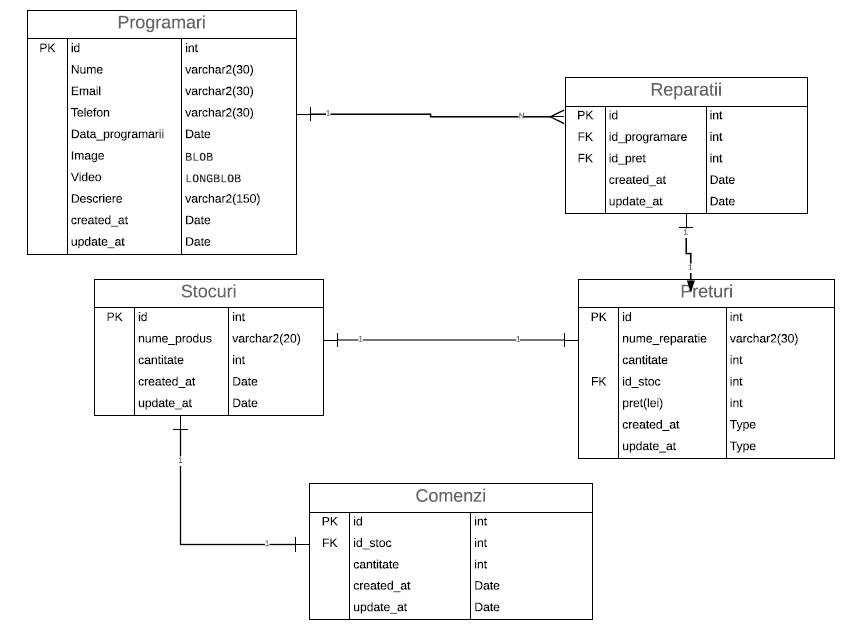
Vom folosi o baza de date relationala pentru a stoca informatiile despre programari, stocuri, comenzi si reparatii.

Baza de date va fi formata din tabelele: Programari, Stocuri, Comenzi, Reparatii, Preturi.

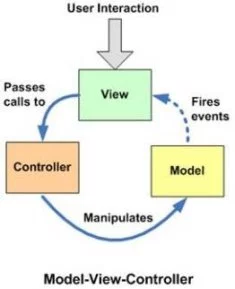
Atunci cand un client face o programare aceasta va fi stocata in tabela Programari.

Adminul va vizualiza programarile, stocurile comenzile si reparatiile din baza de date si le va updata.

Schema E/R a bazei de date:



**3. Design pattern: MVC**



**Views** : view\_programare.php, login.html, view\_programari.php, view\_stocuri.php, view\_comenzi.php, view\_reparatii.php

**Models**: model\_programare.php, authentification.php, model\_programari.php,

model\_stocuri.php, model \_comenzi.php, model\_reparatii.php. Clase: Programare, Stoc, Reparatie, Comanda.

**Controllers**:

controller\_programare.php acesva va lega view\_programare.php cu model\_programare

controller\_ login.php va lega login.html cu authentification.php

controller\_programari.php va lega view\_programari.php cu model\_programari.php

controller\_stocuri.php va lega view\_stocuri.php cu model\_stocuri.php

controller\_comenzi.php va lega view\_reparatii.php cu model\_reparatii.php

Documentat din: <https://herbertograca.com/2017/08/17/mvc-and-its-variants/>

**4. Autentificare**

Vom folosi Json Web Tokens(JWT) pentru a autentifica administratorul. Administratorul va avea un username si o parola pentru a se loga. Admin-ul va fi deja inregistrat cu username-ul si parola sa. El va introduce username-ul si parola in pagina de login. Atunci i se va creea un JWT de forma xxxx.yyyy.zzzz unde prima partea este header-ul, a doua este payload-ul (usr si parola) iar ultima parte este secretul(signature), adica o cheie unic generata pentru user-ul logat. De acum fiecare cerere pe care o va face va contine acest JWT care ii va autoriza accesul pentru resursele cerute. Pe baza acestui JWT administratorul va fi logat pentru 2 ore. Dupa cele 2 ore, signatura va fi stearsa(adica JWT va fi sters de pe server si cel creat la login nu mai fi valid), iar administratorul va trebui sa se logheze din nou.

**5. REST API**

Service-ul va expune un API REST. REST API-ul va putea raspunde la cereri de GET, PUT, POST, DELETE, HEAD, OPTIONS, PATH.(ex. GET appointments/ or GET appointments/{id} ).

Face requesturilor cu ajutorul REST API-ului este posibila doar daca adminul este autentificat(ori contine JWT valid).

**1)GET**

GET appointments/

Protocol: GET

URI: /appointments

Request body: EMPTY

Response:

[

{

"id": 1,

"Nume": "Gigel",

"Telefon": "0757 213 213",

"Email": "[gigel@gmail.com](mailto:gigel@gmail.com)",

"Data": "2008-01-14T04:00:00Z",

"Descriere": "dasdsadasdsadas",

"Imagine/Video":"imagine/video.url",

"created\_at": "2008-01-14T04:33:35Z",

"updated\_at": "2008-01-14T04:33:35Z"

},

{

"id": 2,

"Nume": "Gigel2",

"Telefon": "0757 213 212",

"Email": "[gigel2@gmail.com](mailto:gigel2@gmail.com)",

"Data": "2008-01-14T04:00:00Z",

"Descriere": "dasdsadasdsadas",

"Imagine/Video":"imagine/video.url",

"created\_at": "2008-01-14T04:33:35Z",

"updated\_at": "2008-01-14T04:33:35Z"

}

]

GET appointments/{id}

ex : GET appointments/1

Protocol: GET

URI: /appointments/1

Request body: EMPTY

Response:

{

"id": 1,

"Nume": "Gigel",

"Telefon": "0757 213 213",

"Email": "[gigel@gmail.com](mailto:gigel@gmail.com)",

"Data": "2008-01-14T04:00:00Z",

"Descriere": "dasdsadasdsadas",

"Imagine/Video":"imagine/video.url",

"created\_at": "2008-01-14T04:33:35Z",

"updated\_at": "2008-01-14T04:33:35Z"

}

GET GET/stocks/{id}

ex :GET stocks/1

Protocol: GET

URI: /stocks/1

Request body: EMPTY

Response:

{

"id": 1,

"produs":"frane",

"Cantitate ":"20",

"created\_at": "2008-01-14T04:33:35Z",

"updated\_at": "2008-01-14T04:33:35Z"

}

Similar pentru GET/stocks. Numai ca afisam ca returnam toate stocurile.

De asemenea, un raspuns similar si pentru requesturile GET orders/ si GET orders/1.

**2)POST** - request pentru a adauga campuri noi

POST appointments

Protocol: POST

URI: /appointments

Request body:

{

"Nume": "Gigel",

"Telefon": "0757 213 213",

"Email": "[gigel@gmail.com](mailto:gigel@gmail.com)",

"Data": "2008-01-14T04:00:00Z",

"Descriere": "dasdsadasdsadas",

"Imagine/Video":"imagine/video.url"

}

Response with status 201:

{

meta: {

},

data: {

uri: /appointments/23

}

}

Descriere: Cerere pentru ca programarea sa fie creata. Raspunsul serverului este 201 (creat) cu URI aferent inregistratrarii care este primit ca si raspuns si poate fi folosit in continuare de client.

POST /stocks

Protocol: POST

URI: /stocks

Request body:

{

"produs":"jante",

"cantitate":"5"

}

Response with status 201:

{

meta: {

},

data: {

uri: /stocks/3

}

}

POST /orders

Protocol: POST

URI: /orders

Request body:

{

"produs":"jante",

"cantitate":"5"

"Data":"2008-01-14T04:00:00Z"

}

Response with status 201:

{

meta: {

},

data: {

uri: /orders/5

}

}

**3) PUT** - request pentru a fi modificate sau create campuri

PUT orders

Protocol: PUT

URI: /orders/5

Request body:

{

"id":5,

"produs":"jante",

"Data":"2008-01-14T04:00:00Z"

}

Response with status 412:

{

meta: {

error: 789,

message: 'It must be inserted the "Cantitate" field'

},

data: {

}

}

Description: Requests the book to be modified or created. Server response is 412 (precondition failed) with error code and message describing the problem.

**4)DELETE** - stergerea datelor

Protocol: DELETE

URI: /appointments/2

Request body:Empty

Response with status 202:

{

meta: {

},

data: {

}

}

Description: Requests the book to be removed. Server response is 202 (accepted) meaning that the server accepted the request but the processing has not been completed. In other words, server responded immediately and has the removal pending.

**5)OPTIONS** - informatii despre requesturile posibile pe server