|  |  |  |
| --- | --- | --- |
| **Employee** | **attribute** | **type** |
| **Id** | Uniek id per werknemer. Wordt in upload employee aangemaakt. | Integer,  Default = 0 |
| **Name** | Naam van werknemer, wordt verder niet mee gerekend oid. Zal in de class staan zodat de ***GUI*** deze op kan roepen om weer te geven | String |
| **Wage** | Salaris van een werknemer per uur. Wordt in ***Maluscalc*** (via ***Generator***) aangeroepen om de loonkosten te berekenen. | integer |
| **Weekly\_max\_employee** | Wekelijkse maximum aan shiften wat een werknemer wil werken. Dit is geen waterdichte eis. Als er te weinig mensen zijn moeten er keuzes worden gemaakt. Hoe deze keuze gemaakt gaat worden (doet de manager of het programma dit) staat nog niet vast | integer |
| **Level** | Het level wat iemand is. Dit wordt gebruikt om onderscheid te maken in de ervaring. Twee mensen die ingewerkt worden horen niet samen te staan etc. | Integer, standaard 1 |
| **Task** | De taak die uitgevoerd moet worden. Deze taak staat zowel in de shift als in de werknemer, waardoor een afwasser aan de afwastaak gekoppeld wordt en niet de keuken. | Integer, standaard 1 |
| **Location** | Locatie code om aan te geven om welk filiaal etc het gaat. Wordt niet mee gerekent maar wel gebruikt als constraint | Integer, coffeecompany=1 |
| **Add\_remove\_timeslot** | Lijst met commands die uitgevoerd worden na aanpassingen in app/gui | List of tuples,  Tuple[0]=add or remove  Tuple[1]=list(**employee\_id, week, day, shift, weekly\_max\_employee**) |
| ***Upload\_employee(self)*** | Upload een werknemer in de database. Heeft nu hardcode waar hij zelf op locatie en taak filtert, dit moet later een methode worden. Zet de **name, wage, level, task, location** in een entry en gebruikt de automatisch gegenereerde unieke key om self.id aan te maken |  |
| ***Upload\_availability(self)*** | Upload de **availability** die de werknemer heeft naar de database. De **availability** krijgt de werknemer tijdens het aanmaken vd class binnen en wordt in de init meteen omgezet naar een entry in de database. |  |
| ***Change\_availability(self)*** | Gebruikt de **add\_remove\_timeslot** lijst om **availability** te veranderen. Deze methode moet worden aangeroepen wanneer iemand in de GUI/app klaar is met dingen aanpassen. |  |

|  |  |  |
| --- | --- | --- |
| **Generator** | **attribute** | **type** |
| **Available\_employees** | Een lijst die wordt samengesteld in ***init\_schedule*** waar op index 0 een lijst staat met tuples(**id, wage)** van alle mensen die kunnen werken op de shift die in **shifts needed** op index 0 staat | Lijst met tuples  Tuple[0]= **id**  Tuple[1]= **wage** |
| **Schedule** | Np.array met op elke row een tuple (**week, day, shift, task, employee\_id, index)**. Deze lijst wordt nog redelijk hardcoded aangemaakt (de breedte bijv) dus scalability problemen in toekomst | Lijst met tuples  Tuple[0]= **week**  Tuple[1]= **day**  Tuple[2]= **shift**  Tuple[3]= **task**  Tuple[4]= **employee**  Tuple[5]= **index** |
| ***Init\_schedule(self)*** | Initieert het rooster door eerst **shifts\_needed** te downloaden via ***downloading\_shifts (🡪 queries****)****.*** Vervolgens zet het een tuple (**week, day, shift, task, employee\_id, index)** in **schedule**. Het aanmaken gebeurd nu nog door een np.array te definenen met **len(shifts\_needed)**, columns. ***init\_schedule*** houdt de lijst **available\_employees** bij via ***employee\_per\_shift (🡪 queries*)**. | **Shifts\_needed**=Lijst met tuples  Tuple[0]= **week**  Tuple[1]= **day**  Tuple[2]= **start**  Tuple[3]= **end**  Tuple[4]= **task**  **Columns** = **shifts\_needed** + 1  Want shifts needed heft 5 dingen, waarvan 2 niet nodig zijn en 3 waardes worden nog toegevoegd, dus als er meer waardes toegevoegd worden is die 1 fout |
| ***Improve*** | Roept ***mutate*** aan voor 500x. |  |
| ***Mutate*** | Maakt mutaties aan het schema en controleert of ze de kosten naar beneden hebben gekregen. De kosten worden nu alleen nog maar berekent via het loon. als de verandering niet de kosten naar beneden brengt wordt het teruggedraaid. Het werkt dmv een willekeurige shift te nemen, daar een nieuwe werknemer op te zetten uit de available\_employee en dan de nieuwe kosten uit te rekenen. Zet vorige werknemer terug als kosten hoger worden |  |

|  |  |  |
| --- | --- | --- |
| **Maluscalc** | **attribute** | **type** |
| **Employee\_dict** | Dict met werknemer **name** als key en object als value | dict |
| **Schedule** | **Schedule** zoals beschreven in ***Generator*** | Np.aray |
| **Malus** | Score die nu alleen nog de loonkosten voorstellen en geminimaliseerd moet worden | Int |
| ***Get\_wage\_cost\_per\_week(self)*** | Download de **wages*(🡪 queries*)** en zet de wage van employee 0 standaard op 100000 zodat elk slot altijd gevult zal worden. **Wage\_cost** is een dict die hier wordt gemaakt met [‘schedule’]=totale kosten en [‘week *i*’]=kosten van week *i* |  |

|  |  |  |
| --- | --- | --- |
| **GUI** | **attribute** | **type** |
| **Self.location** | HARDCODED naar 1 (coffecompany) moet uiteindelijk dependend zijn aan de inlog van een gebruiker | int |
| **Self.Com** | Controller class die in de GUI wordt aangemaakt (en daarmee wordt dus ook de thread gestart die de server blijft scannen) | object |

|  |  |  |
| --- | --- | --- |
| **GUI\_employee** | **attribute** | **type** |
| **Self.tasktypes** | Was een **lijst**, is nu een **dict**. Dit is gedaan omdat de db niet met ‘alround’ werkt, maar met 1. Daarom is in de ***init*** de loop ‘for task in **self.takstypes**’ nu veranderd naar **self.tasktypes.keys()**  en wordt nu bij ***\_\_get\_input\_fields(self)*** ipv task, self.tasktypes[task] geappend naar de task lijst. Zie **tasks** voor info! | **Dictionary, was een lijst!** |
| **Self.levels** | Was een **lijst**, is nu een **dict**. Zelfde reden als **tasktypes**. Heb de default status aan manager gegeven (1) maar dit kunnen we ofc veranderen. In de ***init*** wordt nu dan ook over **self.levels.keys()** geloopt. Ook wordt nu bij ***\_\_get\_input\_fields(self)*** de waardes van **self.levels[**currentText()**]** gereturned ipv de currentText() | **Dictionary, was een lijst!** |
| ***\_\_get\_input\_fields(self)*** | Returned **first\_name (str), last\_name (str), wage (int), level (int), tasks (lijst met ints)** |  |
| **controller** | **attribute** | **type** |
| **Self.generator** | Instance van de generator class, wordt in ***main\_GUI*** gemaakt en meegegeven aan ***GUI*** waar uiteindelijk ***controller*** hem krijgt | Class |
| **Self.employee\_list** | Lijst met employee instances, wordt gevuld ***create\_employee*** dus wordt nog niet gevuld met de bestaande employees uit assign. Wordt dus alleen aangevuld als er in de GUI een employee wordt toegevoegd | Lijst |
| **Self.shift\_list** | Lijst met shift instances, zelfde probleem als **self.employee\_list** | Lijst |
| **Self.location** | Locatie code, coffeecompany = 1, krijgt deze code hardcoded uit ***GUI***. | Int |
| **Self.availability** | Stond voorheen in ***generator*** maar staat nu in controller. Is een lijst met codes voor elke shift die elke werknemer kan draaien. | Lijst met tuples  Tuple[0] = **employee.id** (int)  Tuple[1] = **week** (int)  Tuple[2]= **day** (int)  Tuple[3]= **shift** (int)  Tuple[4]= **weekly\_max** (dict) |
| **Self.shifts** | Een lijst met codes voor elke shift die bestaat | Lijst met tuples  Tuple[0]= **week** (int)  Tuple[1]= **day** (int)  Tuple[2]= **start** (str)  Tuple[3]= **end** (str)  Tuple[4]= **task** (int) |
| **Self.close** | Boolean die de thread open houdt. Wanneer ***GUI*** stopt, wordt deze op True gezet waardoor de thread stop | Bool, default = False |
| ***Create\_employee(self, lname, fname, hourly\_wage, level, task)*** | Methode die employees aanmaakt. **Let op!** Hij loopt over **Tasks**. Dat komt omdat **Tasks** een lijst is. Echter kan er per entry in mysql maar 1 waarde worden gezet. Daarom maken we nu in het geval een employee zowel keuken als afwas werkt 2 employees aan. Dit oogt wellicht raar maar zorgt wel voor een administratief duidelijkere database. De employee blijft uniek door zijn **employee.id** | **Tasks** heeft tot nu toe altijd maar 1 waarde, dus de loop veranderd niks. Maar als iemand ooit meerdere dingen kan dan zal er dus een ‘2e’ werknemer worden gemaakt. |
| ***Communicate\_server*** | Loopt altijd op een eigen thread. Zal altijd **self.queue** controleren op nieuwe queries voor de db. Ook vergelijkt hij **self.availability** met de availability zoals in de db beschreven. Als een werknemer dus in de app zijn/haar beschikbaarheid aanpast, zal dit voor de manager op de computer ook duidelijk worden. |  |