Currency system Discord bot

Изготвено от: Александър Ванков

ФН: 97711

Семестър: 5

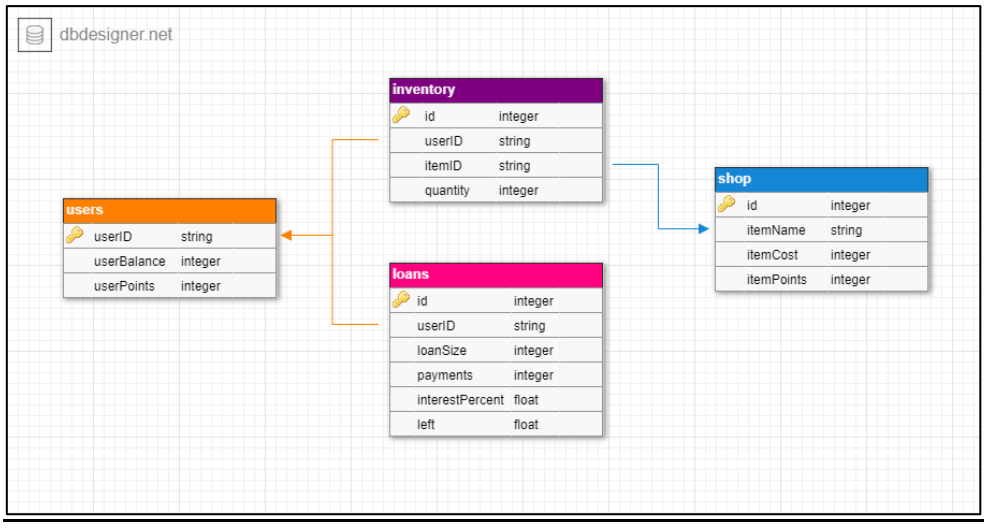
Използвани технологии: NodeJS, JS, GS, SQLite

1. Идея:

Да се направи *Bot* за чат платформата *Discord*. Функционалността на бота е от тип '*currency bot*'. Всеки потребител в дадения *Discord* сървър ще получава 1 монета (*coin*), когато изпрати съобщение. Събраните монети ще може да използва в магазина (*shop*) за да купува предмети, които носят точки. Точките на даден потребител се образуват като към броят на сегашните му монети се прибавят точките от предметите, които е закупил. Всички потребители са включени към ранклиста, според точките, които имат. На всеки 15 минути, ранклистата се качва в таблица в *Google Sheets*. Ранклистата се рестартира всеки месец на 1-во число, като се обявява кои са топ 3 потребители за изминалия месец. Всеки потребител има правото да вземе заем за още монети. Един потребител може да има максимум един активен заем. Всеки ден в 00:00:00 часа автоматично се погасяват всички заеми според наличните монети и предмети на потребителя (по-подробно описание в секция Реализация). По всяко време даден потребител може да провери своя баланс, точки, инвнтар от предмети, предмети в магазина с цени, текущата ранклиста и информация за заема си, ако има такъв.

2. Реализация:

- 2.1 Бази данни:



- 2.1.1 *users*: използва за главен ключ userID, което е *Discord* идентификатора на съответния потребител. В *users* се пази информация и за наличните монети и точки на потребителите. userBalance и userPoitns имат default value = 0.

- 2.1.2 *shop*: пази информация за предметите в магазина и тяхната стойност откъм цена и точки. itemCost и itemPoints нямат default value и не могат да приемат стойност NULL.

- 2.1.3 inventory: Пази информация свързана с инвентара на всеки потребител. userID посочва за кой потребител е съответният запис, itemID посочва за кой предмет от магазина е записа и quantity посочва колко бройки от този предмет притежава потребителя. quantity има default value = 0.

- 2.1.4 *loans*: пази информация за активните заеми. userID посочва за кой потребител е заемът записа. loanSize, payments и interestPercent съответно показват размера на заема, на колко вноски е и лихвения процент. left е сумата, която остава да се изплати за да се погаси заема.

За реализацията на базата се използва *sequelize*. Има 2 файла, освен тези включващи моделите на таблиците, които отговарят за базата. Първият е *databaseInit.js*, който трябва да бъде пуснат отделно веднъж след всяка промяна в структурата на базата за да се инициализира и в *.db* файла. През този файл се задава и съдържанието на магазина, чрез *upsert* метода от *sequalize*. Другият файл е *databaseObjects.js*. В него се създават обекти за всяка от таблиците и се експортират, за да се използват в главното приложение. Също така се дифинират и 2 функции за *User*:

User.prototype.addItem = async function(item) – добавя нов предмет с *quantity* = 1 в инвентара на потребителя. Ако вече има такъв предмет в инвентара, то *quantity* += 1

User.prototype.getItems = function() – функция, която връща инвентара на потребителя

- 2.2 При стартиране на бота:

Декларират се използваните модули: ***discord.js***, ***node-cron***, ***sqlite-to-csv***. Импортират се константи от *config.json* файла: *prefix* и *token*. Импортират се *User*, *Shop* и *Loans* от *databaseObjects.js*. Импортират се *taxValue*, *taxPercent*, *interest* от *loan.json*. Създава се нов *Discord* клиент: **const client = new Discord.Client()** както и *Discord*

колекция: **const currency = new Discord.Collection()**. Целта на колекцията е да улесни работата с базата данни.

Когато ботът е на статус ***'ready'*** :

client.once('ready', async () => { … })

инициализираме *Discord* колекцията с потребителите от *user* таблицата. Създават се и 2 *schedule* събития, чрез **node-cron** модула. Първото се случва всеки ден в 00:00:00 и погасява всички неизплатени кредити. Второто се случва на всеки 15 минути и отговаря за актуализирането на данните в *Google Sheets* таблицата свързана с точките на потребителите.

След всичко това, ботът започва да 'слуша' за съобщения от потребители **client.on('message', async message => { … })**. Ако съобщение е изпратено от друг бот, то то бива игнорирано. Ако съобщение от потребител не започва с *prefix,* то просто се добавя 1 към баланса на потребителя изпратил съобщението. Ако съобщението започва с *prefix*, то ботът извършва съответната команда, ако е коректно написана с коректни аргументи.

- 2.3 Bot команди:

- **help** – Ботът изпраща съобщение във формата на *Discord.MessageEmbed* с изброени всички команди и кратко описание как се ползват и какво правят

- **balance** – Ботът отговаря на потребителя изпратил командата с неговия баланс от монети

- **inventory** – Ботът отговаря на потребителя изпратил командата с неговия инвентар

- **points** – Ботът отговаря на потребителя изпратил командата с неговите точки към момента

- **shop** – Ботът отговаря на потребителя във формата на *Discord.MessageEmbed* с изброени предметите от магазина, цена и колко точки носят

- **buy** – Команда за купуване на предмети от магазина. Приема като аргумент името на предмета, който потребителят иска да закупи.

- **ranking** - Ботът отговаря на потребителя изпратил командата с лист от първите 10 човека в класацията за месеца

- **loan-help** - Ботът отговаря на потребителя във формата на *Discord.Message.Embed* с информация за заемите като цяло и командите свързани с тях

- **get-loan** – Команда за взимане на заем. Приема като аргументи големина на заема и брой вноски

- **make-payment** – Команда за правене на вноска по заем

- **current-loan** - Ботът отговаря на потребителя изпратил командата с информация за текущия му заем, ако има такъв

- 2.4 Методи за currency колекцията:

- **add(id, amount)** – Добавя на потребител (user), с идентификатор *id*, *amount* баланс и точки: **user.userBalance += Number(amount)** и **user.userPoints += Number(amount)**

- **addPoints(id, amount)** – Добавя на потребител (user), с идентификатор *id*, *amount* точки: **user.userPoints += Number(amount)**

- **getBalance(id)** – Връща баланса на потребител с идентификатор *id*, ако няма такова, то връща 0

- **getPoints(id)** – Връща баланса на потребител с идентификатор *id*, ако няма такова, то връща 0

- **setBalanceAndPoints(id, amount)** – Задава на потребител (user), с идентификатор *id*, *amount* баланс и точки: **user.userBalance = amount** и **user.userPoitns = amount**

- 2.5 Реализация на заемите:

Минималанта сума за заем е 150, а минималните вноски са 3. В *loan.json* се задават стойност на таксата (константа), процент такса (според размера на заема), лихва.

async function canAfford(id, size, calcInterest) – булева функция, която определя дали даден потребител може да вземе заем с определени параметри. Сравнват се точките на потребителя + 300 и общата сума, която ще трябва да бъде изплатена за кредита.

Стандартната лихва по кредита се смята по формулата:

лихва + 0.5 * **вноски** / 6, където **лихва взимаме** от *loan.json*, а **вноски** от аргумента, който е въвел потребителя при заявката за заем.

Ако потребител не успее да изплати заема си до края на деня се случва едно от тези две събития:

1. От текущия баланс на потребителя се изважда неизплатената част от заема умножена по 2

2. Ако 1. е невъзможно, то се премахват всички предмети от инвентара на потребителя и неговеите точки стават равни на -200

- 2.6 SQLite-to-CSV и Google Sheets

В началото на index.js файла задаваме кой файл ще преобразуваме и къде да бъде новият файл. В конкретния случай това ще е *database.db*, създаден от *sequelize* и като резултат се получават 5 .csv файла (по един за всяка таблица от базата). Използвайки Google Drive, се прави *backup* на *users.csv* за да може да се използва директно, чрез *Google App Script*.

function importCSVFromGoogleDrive() – взима *users.csv* файла от *Google Drive* и го импортира в *spreadsheet*-а

3. Източници

https://discord.js.org/#/docs/main/stable/general/welcome

https://sequelize.org/

https://libraries.io/npm/sqlite-to-csv

https://nodecron.com/docs/

https://developer.mozilla.org/en-US/docs/Web/JavaScript

https://nodejs.org/en/docs/

https://developers.google.com/apps-script/guides/sheets