**ДЪРЖАВЕН ИЗПИТ**

**ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ –ЧАСТ ПО ТЕОРИЯ НА ПРОФЕСИЯТА**

**Д И П Л О М Е Н П Р О Е К Т**

**Тема: „Разработване на Строител за инструменти за Minecraft“**

**професия: 481020 „Системен програмист“**

**специалност: 4810201 „Системно програмиране“**

**Дипломант: Росен Петков Русев, 12Б клас**

**Ръководител на дипломния проект: Божидар Хинков**

**Подпис: (дипломант)**

**Подпис: (ръководител)**

**София, 2022 г.**

**Съдържание**

[1. Въведение 3](#_Toc1133520434)

[1.1 Уточняване и цел на заданието 3](#_Toc1053386261)

[1.2 Анализ на заданието 3](#_Toc2067145896)

[2. Основна част 4](#_Toc2114075275)

[2.1 Избор на технологии 4](#_Toc1761576657)

[2.1.1 Vue.js 5](#_Toc782503444)

[2.1.2 Fastify.js 6](#_Toc1942641547)

[2.1.3 MongoDB 7](#_Toc2083367289)

[2.2 Сигурност 8](#_Toc1195542495)

[2.3 Оптимизации 10](#_Toc516520652)

[2.4 Затруднения 11](#_Toc378107720)

# 1. Въведение

## 1.1 Уточняване и цел на заданието

Предистория: аз и мои приятели играем популярна игра на име Minecraft с модифицирано съдържание. Една от модификациите (Tinkers Construct v1.16) добавя опция за създаване на персонализиране инструменти и оръжия. Въпросът е кои материали е оптимално да се ползват, какво правят те, защо го правят. Добри ли са? Лоши ли са? Ще можем ли да убиваме дракони или не? Следователно целта на този проект е да достави възможност за бързо и лесно създаване на въпросните инструменти и оръжия.

## 1.2 Анализ на заданието

За да можем да създаваме инструментите, ние трябва да знаем от какво е изграден един инструмент и какво прави всяка част за статистиките на инструмента. След малко разглеждане в книжката, предоставена от модификацията, можем да установим, че един инструмент може да има от 2 до 4 части. Частите се делят на 3 типа, а именно "глава" ("head"), "дръжка" ("handle") и "връзка" ("binding"). Всеки инструмент винаги има поне една глава и поне една дръжка.

Главата дава базови статистики, като издръжливост и щета. Дръжката дава коефициенти към стойностите от главата. Връзката дава само ефекта от материала който е направен. Отделна самият тип на инструмент също дава различни базови статистики и коефициенти.

След малко наблюдения, опити и проверка на източния код на модификацията извлякох следните формули за изчисление:

Durability = avg(head) \* avg(handle) \* tool

MiningSpeed = avg(head) \* avg(handle) \* tool

MiningLevel = max(head)

AttackSpeed = avg(handle) \* tool

AttackDamage = 1 + avg(handle)

Modifiers = head + handle + binding + tool

# Основна част

## Избор на технологии

Изборът на архитектурния модел е MVC. За визуализация на програмата е използвана комбинация от HTML, JavaScript, CSS и работната рамка (framework) Vue.js или Vue. За “бизнес” логиката на приложението е избрана библиотеката Fastify.js имплементирана на TypeScript, което е надградка над JavaScript. За съхраняване на данните се употребява MongoDB.

### Vue.js

Vue (view) е нова и развиваща се работна рамка за уеб пространстовто. Използва се за създаване на динамични и реактивни уеб страници и огромно улесниение при писането на програмен код. Използва се принципа на “компонентите”, което позволява лесно писане, свързване, капсулиране и фокусиране върху дадения компонент. Vue е реактивна рамка, позволявайки при промяна на информация в кода, веднага те да се актуализират на нашата страница.

Пакета "vue-draggable-next": целта на тази библиотека е лесно създаване и контролиране на елементи, които могат да бъдат захванати и завлачени на определените им места на уеб страницата.

Пакета “password-meter”: изчислява сложността на паролата при създаване на нов профил в уеб страницата. Възможните състояния на една парола могат да бъде: много слаба, слаба, средна, силна, много силна и перфектна, като също може да бъде липсваща или твърде къса, но такава порала не се приема. Дадена парола трябва да е с дължина поне 6 символа.

Пакета “axios”: уеб клиент за осъществаване на връзка към сървъра. Предпочетен пред вграденото “fetch”, защото fetch не добавя автоматично глава (header) на уеб заявките спрямо информацията която бива пратена, така и изпращане на жетона за достъп към крайни точки с ограничен достъп.

### Fastify.js

Fastify е уеб сървър писан за node.js създаден за бързина. Предпочетен е пред по-популярна подобен пакет “express”, защото Fastify е по-бърз с 4.7 пъти (https://www.fastify.io/benchmarks/)!

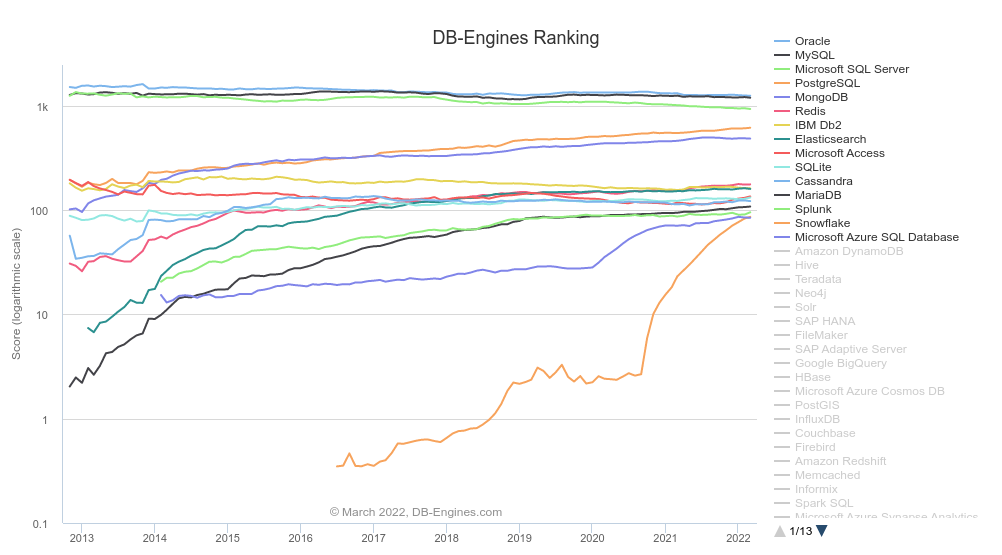
Пакета “bcrypt”: библиотека и стандарт за криптиране на информация. Използвам го за едно-странно криптиране на паролите на потребителите. Добра практика е генерирането на една хеширана парола трябва да отнеме около 270 мили секунди (https://security.stackexchange.com/questions/17207/recommended-of-rounds-for-bcrypt/17238#17238).

Пакета “mongodb”: официалния драйвър пакет предоставен от MongoDB Inc. Предпочетен пред по-популярната си опция “mongoose”, защото предпочитам пълния контрол, който официалния драйвър предоставя.

### MongoDB

MongoDB е технология за запазване на информация под формата на документи или още NoSQL. Разликата между колекциите от документи и таблици от редове в SQL, е че няма връзка между различните колекции, но един документ може да съдържа сложни структури от информация, което на този проект му трябва.

В глобална класация за датабази, MongoDB е на пето място глобално и на първо място като датабаза на принципа на документите (https://db-engines.com/en/system/MongoDB). Това е хубаво, защото е ползвано достатъчно много, че да има редовна поддръжка, но достатъчно рядко ползван, че да няма активни атаки против MongoDB. Тест с мои колеги (без те да знаят използваните технологии) показва, че те се опитват да пробият датабаза базирана на SQL. Това прави запазването и сдобиването на запазената информация много по-сигурно.

.MongoDB е единствената датабаза работеща на документи в най-ползваните петнадесет технологии за запазване на данни.

## Сигурност

При избора ми на различни технологии трябваше да предвидя различните рискове от кибер атаки и различни принципи на защита. Кибер атаки са много и много различни като цел и изпълнение. DDoS (Distributed Denial of Service) атаки се случват непрекъснато и представлява даден сервиз да бъде бомбардиран от големи HTTP заявки, принуждайки го да спре да работи до края на атаката. Такъв вид атаки могат да бъдат предотвратени, като маршрутизатора зад който стои нашия сървър е настроен против DDoS атаки. Други решения са използване на технологии като nginx (Engine X), чиято цел е разпределяне на натоварването на уеб заявките, предотвратявайки спирането на достъп до сървъра или използване на сервизи със същата цел.

Друга заплаха е XSS (Cross Site Scripting). При такъв вид атака целта е да се заложи скрипт на място, където очакваме информация както са описанията на инструментите. За щастие не е нужно да правя нещо неубичайно, тъй като работната рамка която ползва автоматично да изобрази съдържанието автоматично преобразува специалните символи, така че да не могат да навредат на потребителите.

Потребителите трябва да имат профил в платформата, за да могат да запазват и модифицират данни звързни с тях, а именно инструментите, които те създават. Това се осъществява, чрез използването на JWT (JSON Web Token). Това е стандарт за компактно и сигурно прехвърляне и съхраняване на данни. Използва HMAC алгоритъм за криптиране на данни. Фокусира върху подписа на жетона с цел удостоверяване на информацията само от стана на сървъра. Предимството при използване на JWT спрямо бисквитките, е че не е нужна датабаза за помнене на сесии, а самия жетон носи цялата нужна информация спрямо потребителя и се спестява заявка при обръщането на индентификатора на бисквитката в потребителска информация.

## Оптимизации

Понеже сайта е реактивен, това означава че той се зарежда само веднъж, като един пакет (bundle). С него идват и снимките за за компонентите на инструментите. Понеже всеки ресурс ползва текстура на част от инструмент, всеки път като преместим ресурс в място за част се прави заявка за текстурата на всеки сложен ресурс. От друга страна папката с текстури начислява 9.6 МБ. Това за един курс е много, но това първо ни дава достъп до ползване на строителя без нужен достъп до интернет и при прехвърлянето на снимките. От друга стана тези мегабайти са не компресирани и компилирани. След компилиране на клиента общата тежест на проекта е 2МБ, и това е преди сървъра да ги компресира при пращане! Също тези текстури, както и данни за интрументите, не се променят и не е нужно да се съхраняват в датабаза, а веднага се дават с клиента.

## Затруднения

При изобразяването на компоненти и готовия инструмент имах много проблеми. Първият въпрос беше “сега как компонентите знаят от какъв материал са? за коя част? и как взимам графиката?”. Библиотеката която ползвам за влаченене на HTLM елементи позволява да се пренася и информация за елементите. Като се завлачи компонент в някой от контейнерите за строене, към масива за части се добавя обект с id (работната рамка изисква уникални индентификатори при визуализация на масиви в страницата), материала (примерно дърво, камък) и за кой тип компонент става дума. Тази информация, освен че се предава на рамката да я визуализира, се предава на визуализатор, който се сдобива с графиката и я показва. При събирането на графиката трябва да се съобрази, че от типа компонент се нуждаем само от последната дума и да се отдели число ако има такова при някои от инструментите (примерно от tool\_handle\_1 се нуждаем само от handle), да се вземе впредвид типа на инструмента, защото имаше няколко инструмента, чиято структура е по различна от тази на останалите инструменти. Това не беше толкова голям проблем спрямо визуализацията на готов

инструмент. Там използвах платно (canvas) за комбинация на масив от компонентите за готовия инструмент. До сега беше лесно защото визуализирахме предметите, които се използват за инструмента, а не реалните му части, съответно всички графики се взимат от едно място. При взимане на част трябва да се вземе впредвид за какъв тип инструмент става дума, коя част е и ако са повече от една, коя е по поредност, от какъв материал е и дали инструмента е “широк”. Широките инструменти винаги имат повтарящи се компоненти и използват от дебелите дръжки. След много време в тестване успях да открия в кои случаи трябва да се промени начина на взимане на графика при визуализация (примерно при камата, втората част се нарича дръжка (handle), но в пътищата на графиката е гард (guard)). При нормални условия се взима графиката с поредност тип инструмент, тип част и материал. Друга лека спънка е при зареждане на графиката в платното бях забравил да му кажа да изчака самата снимка да зареди преди да я добави.

## Поток на работа

При започване на създаване на инструмент, на потребителя се предоставя достъп до завлачими материали за инструменти, място за

отделните типове части и селектор за типа на инструмента. При всяко завличане на материал му се закача типа част от който е за да си смени графиката подобаващо и се проверява дали всичките места си имат материал. Ако всички имат тогава цялата информация генерирана от контейнерите и материалите в тях се предават на визуализатора за готови инструменти и на калкулатора на статистика. Визуализатора взима графиката за инструмента съобразявайки се с предоставената му информация. Калкулатора взима информацията за всяка част и изчислява всички важни статистики за инструмента (като издръжливост и щета). Когато потребителят е готов с неговия инструмент, може да си го запази при наличието на профил в системата, като трябва да даде име на инструмента и ако иска, описание.

Когато искаме да се сдобием с информация на готови инструменти, може да посети галерията за да разгледа всички инструменти, почвайки от най-новите. Инструментите могат също да бъдат сортирани по тип на инструмента. Там инструментите се визуализират подобно на строителя заедно с име, име на създател и описание ако е дадено.

Може също да посети страницата с профила си, където ще открие само неговите инструменти с опция