**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 4](#_Toc24359)

[1.1 Уточняване и цел на заданието 4](#_Toc24674)

[1.2 Анализ на заданието 4](#_Toc27579)

[2. Основна част 5](#_Toc26943)

[2.1 Избор на технологии 5](#_Toc27643)

[2.1.1 Vue.js 6](#_Toc9251)

[2.1.2 Fastify.js 7](#_Toc25607)

[2.1.3 MongoDB 8](#_Toc12018)

[2.2 Сигурност 10](#_Toc9048)

[2.3 Оптимизации 11](#_Toc10212)

[2.4 Затруднения 12](#_Toc4473)

[2.5 Поток на работа 14](#_Toc22971)

[2.6 Графичен дизайн 15](#_Toc29159)

[3. Схема на работа 16](#_Toc5820)

[3.1 Запазване на инструмент 16](#_Toc24481)

[3.2 Взимане на инструменти 17](#_Toc8309)

[3.3 Обновяване на инструменти 18](#_Toc1635)

[3.4 Схема на потребителите 19](#_Toc10891)

[3.5 Схема на инструментите 20](#_Toc3756)

[3.6 Среда за разработка 21](#_Toc14635)

[4. Пояснение на програмен код 23](#_Toc6907)

[5. Използвани източници 25](#_Toc24531)

[6. Заключение 26](#_Toc26597)

[6.1 Промени спрямо оригиналния план 26](#_Toc19339)

[6.2 Сравнителна характеристика 27](#_Toc28151)

[6.3 Потенциални подобрения 28](#_Toc7114)

[6.4 Затварящи думи 29](#_Toc7036)

# 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.

Избрал съм Vue.js пред неговите две други работни популярни рамки, а именно React и Angular. React го притежават Facebook (сега Meta), което поражда загриженост за сигурността и за поверителността. Angular е твърде абстрактен и корпоразиран с излишни термини и ненужни стандарти, отделно че Google му спряха обновяванията през 31-ви декември 2021 (1).

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

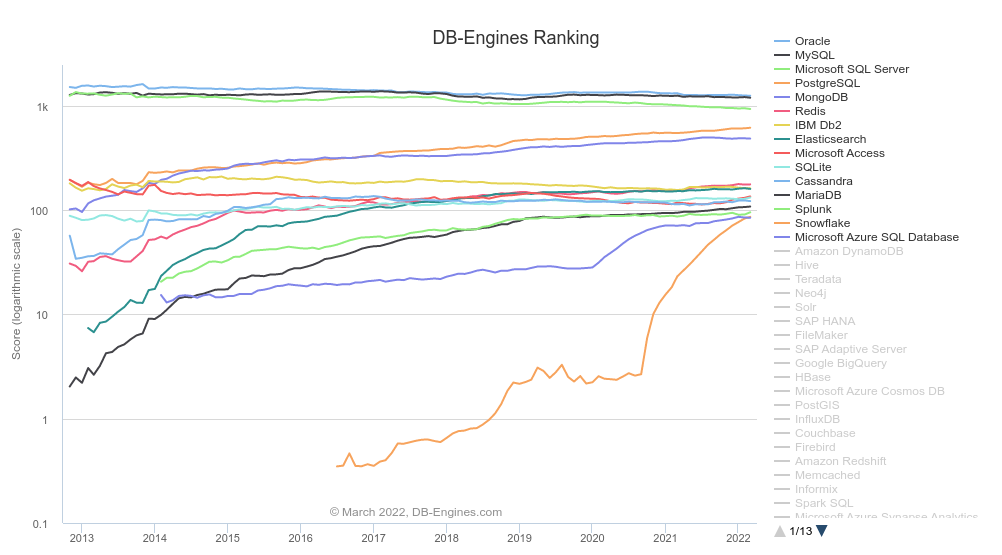
Fastify е уеб сървър писан за node.js създаден за бързина. Предпочетен е пред по-популярна подобен пакет “express”, защото Fastify е по-бърз с 4.7 пъти (2)!

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, потребителят 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 предоставено такова.

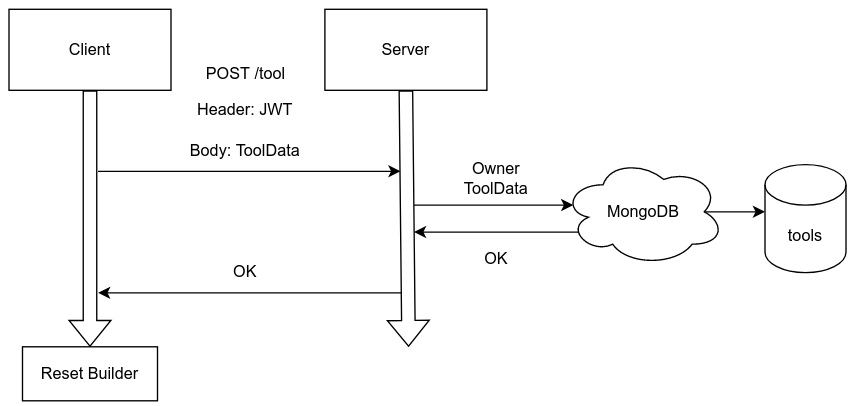
Може също да посети страницата с профила си, където ще открие само неговите инструменти с опция за промяна, която препраща към строителя, зареждайки инструмента на части там, и опция за изтриване на даден инструмент.

## Графичен дизайн

Дизайна на клиента е направен да наподобява графичния интерфейс на играта (Minecraft) като ползва монохромни цветове и квадратни форми. Цветовете на статистиките са подбрани от модификацията за играта (Tinkers’ Construct) директно извлечени от източния му код. Цветовете са съобразени спрямо индекса на достъпност на chromium-базираните браузъри. Дизайна не е предвиден да се ползва от мобилно устройство (портретен позиция на екран), тъй като се предполага, че потребителите имат достъп до настолен компютър.

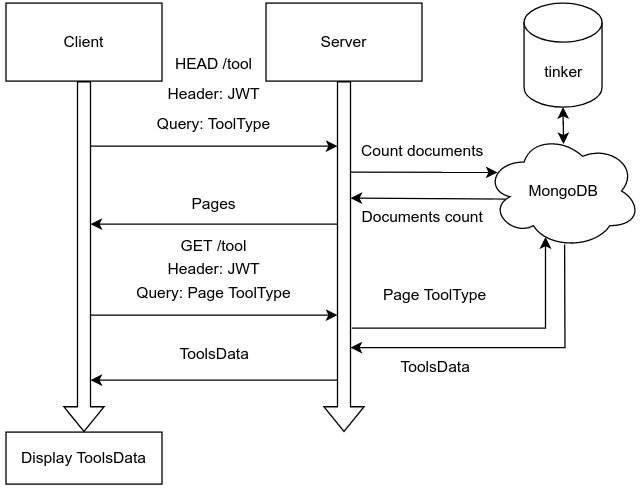
# Схема на работа

## Запазване на инструмент



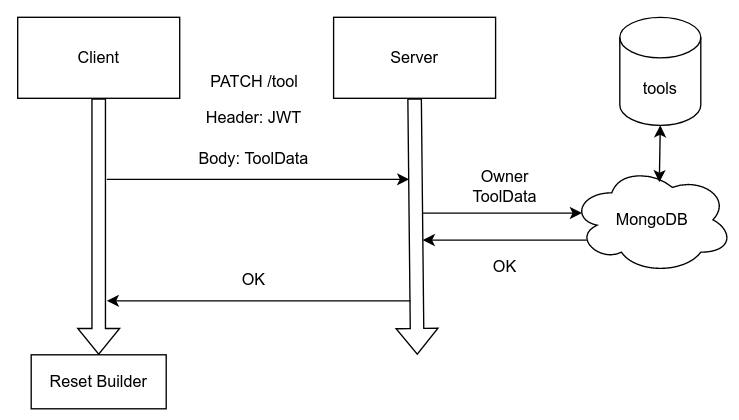
Пояснения: ToolData е цялата информация за един инструмент. JWT е жетона който носи информация за потребителя. Сървъра съединява двете данни и ги запазва в датабазата

## Взимане на инструменти



Пояснения: взимането на инструменти става на два курса за спестяване на малко трафик при смяна на страницата, понеже HEAD заявката казва колко страници инструменти има спрямо броя им делен на инструменти на страница (по подразбиране 5). Така като клиента знае колко страници има не е нужно да пита пак колко страници има, освен когато сменим типа инструмент. Ако потребителя е на своя профил при заявките се закача и неговия индентификатор.

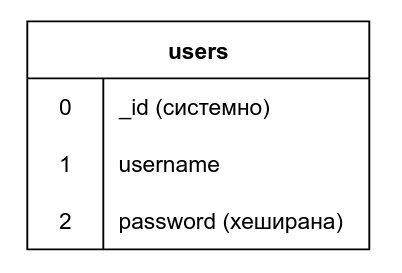
## Обновяване на инструменти



Пояснение: много подобно на създаване на инструмент с разликата че вместо да се запише нов документ, се проверява кой документ съвпада с потребителския индентификатор и инструментния индентификатор.

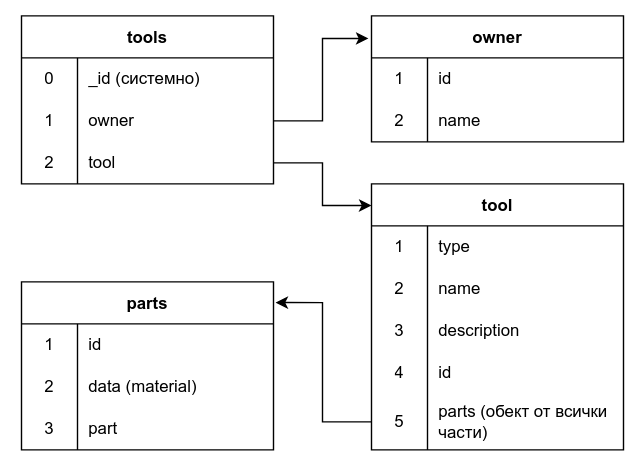
Така имаме всички CRUD операции изпълнени.

## Схема на потребителите



Основна информация за всеки регистриран потребител, а именно негов прякор и едностранно криптирана парола по bcrypt алгоритъма.

## Схема на инструментите



Малко по-сложна структура за самите инструменти. Записваме информация за притежетеля на даден инструмент и самия интрумент. За притежателя му записваме прякора и индентификатора. За инструмента му записваме какъв тип инструмент е, името и потенциалното му описание, индентификатор и карта на частите му. Частите са записани с “клиентски индентификатор”, който е нужен на работната рамка, информация за материала на частта и какъв тип част е (примерно малка дръжка).

## Среда за разработка

Една добра програма е резултат на кадърни разработчици и адекватна среда за самата разработка. Използвал съм Visual Studio Code като среда за разработка, която е единствената среда която ползвам за всеки мой проект. Лека, удобна и модуларна (като инструментите) аз мога да си я настроя точно така както ми трябва да работи. С хиляди и дори милиони разширения всичко може да се направи във VSCode (включително да се разглеждат протеини (3)). Разширения които аз съм ползвал и са помогнали в създаването на този проект са:

Vetur: официално разширение за VSCode за добавяне на изтъкване на синтаксиса и семантиката на Vue, както и Emmet, IntelliSense, Linting и Debug инструменти. Всичко това значително улеснява работата с работната рамка.

Todo Tree: лесно проследяване на части от код които трябва да бъдат имплементирани или завършени.

Auto Rename Tag: при промяна на HTML етикет, разширението променя затварящия етикет подобаващо.

CodeSnap: инструмент за правене на снимки на програмен код, използван за пояснение на кода по-късно в книжното тяло.

Vue.js devtools: официално разширение за браузър. Показва подробна информация за в моментното състояние на страницата, включително DOM дървото и стойностите на всеки компонент.

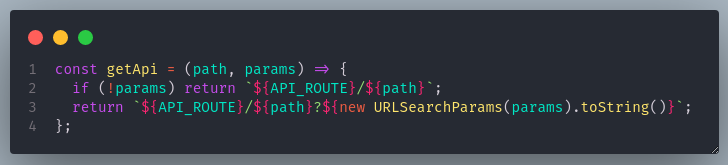
MongoDB Compass: официалното приложение на MongoDB за разглеждане на датабази базирани на MongoDB. Значително по-бързо и удобно от уеб версията си.

pnpm: инструмент за инсталиране на node.js пакети. Избран е заради неговата бързи и ефикасност на дисковото пространство. Когато се зададе команда за сваляне на даден пакет, pnpm го запазва в склад намиращ се в корена на в моментния диск и прави връзка с проекта. Така всеки следващ проект използва същия този склад и не е нужно нито да се теглят на ново, нито да съществуват всеки път в нов проект, правейки спестяването на място експоненциално по-ефективно! Освен зависещите пакети на моя проект, pnpm свързва зависещите пакети с техните зависещи. Така става много бързо и много ефикасно.

draw.io: инструмент за създаване на диаграми и схеми, с който са изработени схемите в точка 3.

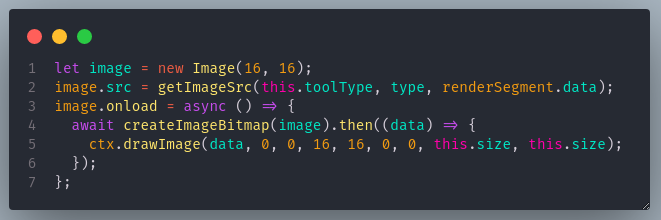
# Пояснение на програмен код

Използвам програмния стил, който разширението “prettier” поддържа, а именно началото на кодов блок започва на същия ред като дефиницията му, две спации на ниво вместо табулация и единични кавички. Стрелкови функции (част от ES6 стандарта) при дефиниция на функции. Тесктови шаблони при вмъкване на променливи в текст. Ранни връщания за избягване на ползването на else ключовата дума (има 7 else оператора в целия код, като 4 от тях са част от HTML, което го ползва работната рамка за визуализиране на елементи)



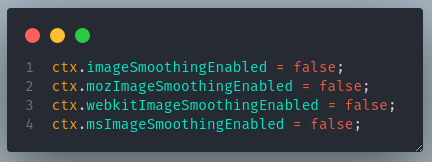
Докато правя заявка към сървъра, аз трябва да знам къде се намира той. Имам функция, която се вика всеки път преди изпращането на дадена заявка. Функцията взима глобално дефиниран път от средата и

към него добавя крайната точка като аргумент и ако има параметри ги добавя и тях.



При генериране на път към изображението при поток на информацията, всеки път се предава към чертаещета функция (render). Функцията извиква от клиента пътя към снимката и взима балон (blob), който съдържа base64 криптирана версия на снимката на дадена част на инструмента. Всяка снимка на част е 16 на 16 пиксела прозрачна снимка, която се добавя към платното (canvas) за готови инструменти. При добавянето на платното размера се разширява по дадена променлива, за да може ясно да се чете от потребителя в зависимост в какъв контекст е самата снимка (Строителя и Галерията имат различни размери на завършения инструмент). при такава операция се получава ефекта на размазване което универсално от

браузърът се оправя със следния код (нужно е да се изпълни само веднъж, за да вземе постоянен ефект):



# Използвани източници

Играта от където идва идеята - Minecraft <https://minecraft.net>

Tinkers’ Construct

<https://curseforge.com/minecraft/mc-mods/tinkers-construct>

Vue.js <https://vuejs.org>

Vue router <https://router.vuejs.org>

Fastify <https://www.fastify.io>

MongoDB <https://www.mongodb.com>

VS Code <https://code.visualstudio.com>

npm <https://www.npmjs.com>

pnpm <https://pnpm.io>

draw.io <https://draw.io>

(1) <https://blog.angular.io/discontinued-long-term-support-for-angularjs-cc066b82e65a>

(2) <https://www.fastify.io/benchmarks/>

(3) <https://marketplace.visualstudio.com/items?itemName=ArianJamasb.protein-viewer>

(4) <https://www.electronjs.org>

(5) <https://docs.google.com/spreadsheets/d/1g40JSmZNyav8-qYI3owxA1IvEIhI10ebs9QAoPpGS4g/edit#gid=402220955>

# Заключение

## Промени спрямо оригиналния план

Оригинално исках да ползвам SQL с knex библиотека за запазване на инструментите в датабаза, но прецених че ще ми отнеме излишно много време и усилия да измисля заявките, да “нормализирам” модела на инструментите и да защита всяка заявка от атаките на “SQL Injection”. За сметка на това MongoDB запазва информацията във форматав, в която така или иначе се ползва от клиента и сървъра спестявайки обръщането от бинарен код в JSON документ.

Идеята беше клиета да бъде уеб базирано настолно приложение, имплементиращо electron.js (4). Проблемът е че играта Minecraft с много модификации изисква много RAM памет на системата и се предполага, че вече има отворен браузър, в който най-вероятно се

използват други инструменти и/или инструкции за другите модификации за играта. Electron използва браузър с отворен код “chromium”, за да може да зареди HTML страница като настолно приложение. Проблемът е, че това отваря напълно нова инстанция на браузър, ефективно удвоявайки общото RAM потребление на вече съществуващ отворен браузър.

## Сравнителна характеристика

Понеже проекта се фокусира върху много специфична работа, няма много подобни проекти, които да изпълняват тази функция. Това което открих да има е една excel таблица (5), която е имплементация за версия 2.5.6, а моята е за версия 3.4.2.60. ОГРОМНА разлика във версиите и в цялата логика на създаване, калкулиране и типове на инструментите. Други инструменти НЕ съществуват или са далеч от това да работят

.

## Потенциални подобрения

В момента имаме само инструменти, но в модификацията идват и още няколко типа предмети, като брони и черепи. Те също дават някакви статистики подобни на вече съществуващите. Обаче броните и черепите са нови за версия 3 на модификацията, до сега във версия 2 нямаше и двете, но имаше други типове инструменти и материали (важно да отбележим че всички версии все още се ползват активно). Модификацията може да бъде добавена в тематичен пакет от модификации и статистиките на материалите и на частите могат да бъдат променени с цел подобряване на баланса на играта или създаване на определено усещане или история. Имайки впредвид тези неща ще е добре да се добави опция за смяна на версията и всички материали и инструменти, които зависят на версията да бъдат обновени, а за пакетите, промените от тях да бъдат добавени в ново поле на даден материал или част, така че да презапише нейните стойности. За версия 2 има и отделна модификация, която добавя и броните с пак много нови неща, които да бъдат имплементирани от нашата система. Ще се създаде и възможност на потребителите да създават и колекции от инструменти, брони и други модуларни

предмети, връзвайки към тях даден пакет или ръчно изработени модификации по статистиките и маркиране на кои модификации са използвани за дадена колекция или за самостоятелен инструмент. Галерията може да и се добави опцията за запазване на инструменти като “любими” или гласуване и съответно сортиране по това, колко даден инструмент или колекция е харесвана или запазена като любима от други потребители на платформата. Тези модуларни предмети изискват и някакви материали за създаването им, за което всъщност имаме този инструмент, да избегнем разхода на материали и време при създаването им. Потенциал може да се пренапише и на компилационнен език с цел на реално настолно приложение без притеснението от недостик на RAM паметта.

## Затварящи думи

Ура! Вече имаме използваем инструмент за създаване на модуларните инструменти на Tinkers’ Construct, който ни помага при избор на създаване на инструмент за позлване в играта. Направено лесно и достатъчно сигурно за да се ползва без безспокоението нещо лошо да се случи.