

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Webový 3D simulátor těles ve vesmíru

Michal Struna

Bakalářská práce

2019

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne ?. ?. 2019

Michal Struna

Poděkování

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

ANOTACE

Bakalářská práce se v praktické části zabývá implementací webové aplikace pro 3D vizualizaci těles ve vesmíru. V rámci aplikace je kladen důraz na dynamický obsah, na kterém se mohou všichni uživatelé po úspěšné autentifikaci podílet. Veškerý obsah je možné pomocí serverového REST API upravovat. Teoretická část je zaměřena na popis technologií pro tvorbu webových aplikací, vysvětlení životního cyklu aplikace a uvedení řešení několika problémů spojených s implementací. Součástí teoretické části je také detailní popis všech stránek nacházejících se v aplikaci.

KLÍČOVÁ SLOVA

webová aplikace, simulátor, vesmír, astronomie, TypeScript, 3D

TITLE

Web 3D simulator of bodies in universe

ANNOTATION

The bachelor thesis in practical parts deals with implementation of web applications for 3D visualization of bodies in space. Within the application, emphasis is placed on the dynamic content on which all users can participate after successful authentication. All content can be edited using the REST API. The theoretical part focuses on the description of technologies for creating web applications, explaining the life cycle of the application and introducing solutions to several problems connected with implementation. Part of the theoretical part is also a detailed description of all the pages that are in the application.

KEYWORDS

web application, simulator, universe, astronomy, TypeScript, 3D

OBSAH

Seznam obrázků	8
Seznam zkratek	10
Úvod	11
1 3D simulace a jejich řešení	12
1.1 Principy počítačové 3D grafiky	12
1.1.1 Pixel, Voxel a Shader	12
1.1.2 Modelování	12
1.1.3 Texturování	12
1.2 WebGL	12
1.2.1 Grafický řetězec	12
1.2.2 Shader	12
1.2.3 Kontext	13
1.3 Simulace a emulace	13
2 Použité technologie	14
2.1 Jazyk TypeScript	14
2.2 Knihovna React	14
2.2.1 Syntaxe TSX	14
2.3 Knihovna Three.js	15
2.4 Framework Node.js	15
2.5 Databáze MongoDB	15
2.6 Knihovna Mongoose	15
2.7 Nástroj Swagger UI	16
2.8 Nástroj Webpack	16
2.9 Verzovací systém Git	17
2.10 Balíčkovací systém NPM	17
2.11 Preprocesor SASS	17

3	Návrh a vývoj aplikace	18
3.1	Struktura projektu	18
3.2	Uživatelské rozhraní	19
3.2.1	Akce	21
3.2.2	Reducer	21
3.3	3D grafika	22
3.3.1	Těleso	23
3.3.2	Orbita	24
3.3.3	Světlo	24
3.3.4	Prstence	25
3.3.5	Popisek	25
3.3.6	Částice	25
3.3.7	Omezení	27
3.4	Serverová část	27
3.4.1	Zachycení HTTP požadavku	27
3.4.2	Zpracování HTTP požadavku	28
3.5	Databáze	29
3.5.1	Připojení do databáze	30
3.5.2	Mongoose schéma	30
3.5.3	Mongoose plugin	31
3.5.4	Základní práce s databází	32
3.5.5	Aggregation Framework	33
4	Rozvržení aplikace	34
4.1	Hlavní stránka	34
4.2	Autentifikace uživatele	34
4.2.1	Identita	34
4.2.2	Přihlášení	35
4.2.3	Registrace	35
4.2.4	Zapomenuté heslo	36
4.2.5	Reset hesla	36
4.3	Uživatel	36
4.3.1	Detail uživatele	36

4.3.2	Editace uživatele	37
4.4	Simulátor	37
4.5	Uživatelský panel	38
4.5.1	Přehled	38
4.5.2	Seznam těles	39
4.5.3	Detail tělesa	40
5	Problémy řešené při implementaci	43
5.1	Omezení viditelnosti těles	43
5.2	Sestavení aplikace a optimalizace	44
5.2.1	Transpilace a sloučení JS a CSS souborů	44
5.2.2	Minifikace a GZIP komprese	45
5.3	Bezpečnost a ochrana dat	45
5.3.1	Autentizace pomocí tokenů	45
5.3.2	Hashování hesel	46
5.4	Zobrazování hodnot fyzikálních veličin	46
5.5	Vykreslování časové osy	48
	Závěr	49
	Použitá literatura	50
	Seznam příloh	52
	Seznam souborů zdrojových kódů na přiloženém nosiči	58

SEZNAM OBRÁZKŮ

Obrázek 1	Životní cyklus architektury Redux	20
Obrázek 2	Jednoznačná definice tělesa	23
Obrázek 3	Jednoznačná definice orbity tělesa	24
Obrázek 4	Jednoznačná definice prstence tělesa	25
Obrázek 5	Hlavní pás planetek vygenerovaný pomocí částic	26
Obrázek 6	Hlavní stránka	34
Obrázek 7	Formulář pro zjištění identity uživatele	35
Obrázek 8	Formulář pro přihlášení uživatele	35
Obrázek 9	Formulář pro registraci uživatele	36
Obrázek 10	Simulátor	38
Obrázek 11	Přehled	39
Obrázek 12	Seznam těles	40
Obrázek 13	Detail tělesa	41
Obrázek 14	Časová osa tělesa	41
Obrázek 15	Diskuse o tělese	42
Obrázek 16	Sestavení aplikace a optimalizace	44

SEZNAM ZDROJOVÝCH KÓDŮ

Zdrojový kód 1	Porovnání JS a JSX	14
Zdrojový kód 2	Konfigurace nástroje Webpack	16
Zdrojový kód 3	Ukázka práce s NPM	17
Zdrojový kód 4	Správné a nesprávné použití importu souboru z modulu	19
Zdrojový kód 5	Ukázka architektury Redux	20
Zdrojový kód 6	Redux akce za využití vlastní knihovny	21
Zdrojový kód 7	Redux reducer za využití vlastní knihovny	21
Zdrojový kód 8	Práce s vlastní knihovnou pro 3D grafiku	22
Zdrojový kód 9	Výpočet geometrie orbity tělesa	24
Zdrojový kód 10	Naplnění geometrie částicemi ve tvaru koule	26
Zdrojový kód 11	Definice cesty v REST API	27
Zdrojový kód 12	Zpracování HTTP požadavku	28
Zdrojový kód 13	Automatizovaná tvorba modelových tříd	29
Zdrojový kód 14	Připojení k databázi v aplikaci	30
Zdrojový kód 15	Vytvoření databázového schématu	30
Zdrojový kód 16	Vytvoření a použití databázového pluginu	31
Zdrojový kód 17	CRUD operace nad kolekcí uživatelů	32
Zdrojový kód 18	Hashování hesel v Mongoose schématu	46
Zdrojový kód 19	Ukázka formátování jednotek	47
Zdrojový kód 20	Ukázka zaokrouhlování hodnot	47
Zdrojový kód 21	Tvorba vlastních jednotek	47
Zdrojový kód 22	Příklad použití komponenty EventsArea	48

SEZNAM ZKRATEK

3D	Three Dimensional
API	Application Programming Interface
BSON	Binary JSON
CSS	Cascading Style Sheets
CRUD	Create, Read, Update, Delete
DOM	Document Object Model
ES	ECMAScript
FPS	Frames Per Seconds
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HW	Hardware
JPEG	Joint Photographic Experts Group
JS	JavaScript
JSX	JavaScript XML
JSON	JavaScript Object Notation
MVC	Model View Controller
PNG	Portable Network Graphics
REST	Representational State Transfer
SASS	Syntactically awesome style sheets
TS	TypeScript
TSX	TypeScript XML
UI	User Interface
URI	Uniform Resource Identifier
WebGL	Web Graphics Library
XML	Extensible Markup Language

ÚVOD

V dnešní době zažívá odvětví astronomie velký rozmach. Lze nalézt mnoho portálů, které nově objevené informace ihned zveřejňují. Často se však jedná o rozsáhlé monografie, které laikovi příliš neřeknou. Navíc ne vždy jsou dostupné v české lokalizaci. Na druhé straně existují 3D simulátory, které je ale nutno stahovat z internetu a poté instalovat. Tyto simulátory však obsahují pouze minimum informací a spíše než informační prostředek a komunitní portál slouží jako pouhá vizuální scéna.

Cílem této bakalářské práce, je vytvořit aplikaci, která poskytne jednoduchý pohled na astronomii lidem, kteří by se o tomto odvětví něco rádi dozvěděli. Díky obsáhlé databázi dat však nabízí i pohodlný a dostupný zdroj informací pro pokročilejší uživatele. Spojuje tak textové zdroje a grafické aplikace. Uživatel má možnost si libovolnou vlastnost libovolného tělesa zobrazit graficky před sebou a tuto vlastnost pak porovnat napříč všemi tělesy v databázi. Vše je dostupné v české lokalizaci a zároveň je celá aplikace online.

Obsah webové aplikace je plně dynamický a kdokoliv do něj může přispívat svými znalostmi. Všechny takto přidané informace však prochází schvalovacím procesem, kterého se účastní administrátor.

1 3D SIMULACE A JEJICH ŘEŠENÍ

1.1 Principy počítačové 3D grafiky

1.1.1 Pixel, Voxel a Shader

1.1.2 Modelování

1.1.3 Texturování

1.2 WebGL

WebGL je verze *OpenGL* určená pro web. Jedná se o javascriptové rozhraní pro zobrazování nativní 2D a 3D grafiky. [3] Program pracující s *WebGL* se skládá z javascriptového řídicího kódu a z tzv. *shaderu*.

1.2.1 Grafický řetězec

Každý pixel prochází před vykreslením několika fázemi, které ovlivňují jeho výslednou barvu. Příkladem může být stínování, osvětlení nebo hloubkový test. Tyto fáze dohromady tvoří grafický řetězec (*graphics pipeline*). [3]

Dříve (ve verzi *OpenGL 1.1*) existoval pouze *fixed-function pipeline*. Ten umožňoval jednotlivé fáze povolit nebo zakázat, ale neumožňoval je upravovat. Později (*OpenGL 2.0*) byl představen programovatelný grafický řetězec. Programátor tedy může jednotlivé fáze nahradit svým vlastním programem. Tento program se nazývá *shader*. [3]

1.2.2 Shader

Shader, je program určený přímo pro grafický procesor. Je napsán v *OpenGL Shader Language*, který vychází z jazyka C a do velké míry přebírá i jeho syntaxi. Samotný *shader* se skládá z navzájem oddělených programů *vertex shader* a *fragment shader*. Je možné je umístit do samostatného souboru nebo jako textový řetězec do hlavního programu. [3]

Oba programy mají vlastní funkci `main`, která je vstupním bodem. *Vertex shader* se provede nad každým vrcholem grafického primitiva. Vstupem i výstupem je vždy jeden vrchol. Nelze tedy přidávat nebo odebírat vrcholy. Je možné pouze uplatňovat různé operace, např. transformace, na již existující vrcholy. *Fragment shader*, někdy také *pixel*

shader, je prováděn nad každým pixelem grafického primitiva. Vedle těchto programů jsou zde obsaženy i některé původní fáze z *fixed-function pipeline*. [3]

1.2.3 Kontext

Pro práci s *WebGL* v prohlížeči je třeba získat tzv. grafický kontext. To je v tomto případě javascriptový objekt implementující rozhraní *WebGL*. Pro tento účel existuje metoda `canvas.getContext('webgl')` dostupná na *HTML* elementu `canvas`. [3]

1.3 Simulace a emulace

Podstatou simulace je nahrazení zkoumaného dynamického systému jeho simulujícím systémem za účelem zjistit o původním systému nějaké informace. [18]

Vedle simulací je možné se setkat i s emulacemi. U emulace jsou naopak všechny informace o původním systému známy. Emulující systém pouze umožní práci s původním systémem, který nemusí být k dispozici.

Podle těchto vlastností lze na aplikaci, jež je součástí této práce, nahlížet jak jako na simulátor, tak i na emulátor. Uživatel může využít simulaci těles pro to, aby se mimo jiné dozvěděl vzájemnou polohu či rychlost v určitém čase.

Na druhou stranu, aplikace se řídí Keplerovými zákony. Ty říkají, jak se pohybují tělesa po eliptických drahách, ale neříkají proč se tak děje. [1] Aplikace tedy reálně nepracuje se skutečnými fyzikálními vztahy těles, pouze se řídí podle pravidel, která vyplývají z Keplerových zákonů.

2 POUŽITÉ TECHNOLOGIE

2.1 Jazyk TypeScript

TypeScript je programovací jazyk vyvinutý firmou *Microsoft*. Jedná se o nádstavbu jazyka *JavaScript*, která přidává statické typování a další vlastnosti objektového programování. Kód napsaný v jazyce *JavaScript* je kompatibilní s *typescriptovým* kódem. Pro kompatibilitu v prohlížečích je nutné veškerý *TypeScript* transpilovat do *javascriptového* kódu. [7]

2.2 Knihovna React

Javascriptová knihovna *React*, jejíž autorem je *Facebook*, usnadňuje a zefektivňuje tvorbu *UI*. [6] Přináší tzv. *one-way data binding*, které zaručuje okamžitou aktualizaci *UI* při změně stavu aplikace. [8] *React* si vytváří vlastní virtuální *DOM*, který je na rozdíl od toho v *HTML* rychlejší. V tomto modelu pak *React* provádí všechny své operace. Teprve když je třeba provést změnu v prohlížeči, je třeba aktualizovat *HTML*. [6]

2.2.1 Syntaxe TSX

Vytvářet zanořovací strukturu komponent může být nepřehledné. Proto se často s knihovnou *React* používá i syntaxe *JSX*. Ta umožňuje psát *javascriptový* kód v podobě *XML* tagů. *JSX* syntaxi je z důvodů kompatibility prohlížečů nutné transpilovat do nativního *JavaScriptu*. [6] *TSX* je pak obdoba *JSX* v jazyce *TypeScript*.

```
React.createClass('MyComponent', {  
  children: 'Hello ' + name + '!',  
  className: 'block'  
})
```

```
<MyComponent className='block'>  
  Hello {name}!  
</MyComponent>
```

Zdrojový kód 1: Porovnání JS a JSX

2.3 Knihovna Three.js

Three.js je javascriptová knihovna usnadňující práci s *WebGL*. To je rozhraní, které umožňuje přístup k *HW* komponentám počítače specializovaných pro zpracování grafiky 2D a 3D grafiky přes *JavaScript*. Veškerou tuto grafiku je možno vykreslovat v prohlížeči za využití elementu `canvas`. [3]

2.4 Framework Node.js

Framework *Node.js* umožňuje používání jazyka *JavaScript* na serveru. Pracuje pouze s jedním vláknem a funguje na asynchronním neblokujícím zpracování požadavků. Jakmile je dokončen požadavek, jeho callback uvedený v argumentu se zařadí do fronty. Tzv. *event loop* pak zjišťuje, zda je zásobník zpracovávaných operací prázdný. Pokud ano, vloží do něj první callback z fronty. Tento cyklus se opakuje po celou dobu běhu serveru. [9]

2.5 Databáze MongoDB

MongoDB je *NoSQL* multiplatformní dokumentová databáze s otevřeným zdrojovým kódem. Narozdíl od relačních databází používá dokumenty ve formátu *BSON*, což je binární obdobou *JSON*. Při vytváření dokumentů si databáze automaticky vytváří vlastní unikátní ID. Uložené dokumenty lze mimo jiné vyhledávat na základě prvků v poli, podle rozsahu nebo podle regulárních výrazů. V *MongoDB* je možné indexovat libovolné pole. Indexy jsou koncepčně podobné, jako v relačních databázích. [10]

2.6 Knihovna Mongoose

Knihovna *Mongoose* pro *JavaScript* zjednodušuje práci s *MongoDB*, zejména pak vytváření schémat a validaci dat. Umožňuje práci s pluginy, což jsou uživatelské funkce, které po zaregistrování na daném schématu mohou zautomatizovaně provádět předem definované činnosti nebo reagovat na různé události. [10]

2.7 Nástroj Swagger UI

Swagger UI je nástroj pro vizualizaci specifikace *Open API*. Vygenerovaný interaktivní webový dokument umožňuje vytvářet *HTTP* požadavky na cílové rozhraní a zobrazit výsledek těchto požadavků. [17]

2.8 Nástroj Webpack

Prohlížečový *JavaScript* nativně nepodporuje rozdělování kódu do modulů (jako např. *Java* do balíčků nebo *C++* do jmenných prostorů). Jediným způsobem je importovat do *HTML* větší množství *javascriptových* souborů pomocí tagu `<script>`. Tento postup je ale špatnou praktikou a na web má negativní dopady. [13]

Díky nástroji *Webpack* je možné v *JavaScriptu* využívat modulární systémy jako *CommonJS*, *AMD* nebo *ES modules*. Je tak možné větší množství souborů propojit pomocí klauzulí `require` nebo `import`. [12]

Webpack také umožňuje využívat tzv. *loadery* třetích stran. To vede k možnosti podobně načítat, slučovat či parsovat i soubory jiných typů, např. *CSS*, *SVG* nebo *JSON*. Zároveň za využití bohaté nabídky pluginů lze všechny tyto soubory modifikovat, např. převádět novou *JavaScript* verzi *ES6* na starší, podporovaný prohlížeči. [12]

```
{
  entry: './src/index.js',
  output: { filename: 'index.min.js' },
  module: {
    rules: [
      { test: /\.tsx?$/, use: ['ts-loader', 'babel-loader'] }
    ]
  }
}
```

Zdrojový kód 2: Konfigurace nástroje Webpack

2.9 Verzovací systém Git

Git je systém správy verzí vytvořený *Linusem Torvaldsem*. Při vytváření nové verze dat vytvoří snímky všech souborů tak, jak v daný okamžik vypadají a na tyto snímky pak uloží reference. V případě, že se soubor nijak nemění se nevytváří nový snímek, ale pouze se nastaví reference na ten předchozí, který je identitický. [11]

Pro spravované soubory používá tři stavy. Při změně souboru se nastaví jeho stav na *modified*. Stav *staged* znamená, že soubor byl označen k tomu, aby byl zapsán v další verzi. Zapsaný soubor je ve stavu *committed*. Všechny tyto změny jsou však pouze lokální. Pro umístění změn do vzdáleného repozitáře je třeba všechny zapsané soubory odeslat (*push*). Ostatní s přístupem do repozitáře si pak mohou tyto změny stáhnout (*pull*). [11]

2.10 Balíčkovací systém NPM

NPM je správce balíčků pro serverový i klientský *JavaScript*. Řídícím souborem v projektu je *package.json*, který obsahuje informace o projektu a určuje, které balíčky jsou součástí projektu. Všechny balíčky jsou umístěny do adresáře *node_modules* v projektu. Součástí je také soubor *package-lock.json*, který zachovává verze nainstalovaných balíčků. [5]

```
npm update // Update packages.
npm install // Install existing dependencies.
npm search mongo // Find out name of package.
npm install --save mongodb // Add new dependency.
npm uninstall --save-dev webpack // Remove dev dependency.
npm list // Show dependency tree.
```

Zdrojový kód 3: Ukázka práce s NPM

.

2.11 Preprocesor SASS

Preprocesor *SASS* přidává do *CSS* možnost zanořování selektorů, proměnné, funkce, podmínky a další vlastnosti zlepšující čitelnost kódu. Pro kompatibilitu s prohlížeči je nutné ho transpilovat do *CSS*. [14]

3 NÁVRH A VÝVOJ APLIKACE

3.1 Struktura projektu

React nepřichází s žádnou doporučenou strukturou projektu. [15] Proto došlo k vytvoření vlastní struktury. Vzhledem k velkému množství souborů je celý obsah projektu členěn do adresářů a modulů.

- **dist**: Produkční sestavení aplikace. Veškerý obsah adresáře se generuje automaticky.
- **node_modules**: Externí knihovny pro serverovou část.
- **src**: Zdrojové soubory aplikace.
 - **Client**: Podprojekt, který obsahuje všechny klientské soubory.
 - * **node_modules**: Externí knihovny pro klientskou část.
 - * **src**: Zdrojové soubory klientské části.
 - **Controls**: Modul obsahující ovladače a tlačítka využitá v menu.
 - **Forms**: Modul umožňující tvorbu formulářů.
 - **Global**: Hlavičkové soubory dostupné v globálním prostoru.
 - **Panel**: Modul pro panel tělesa.
 - **System**: Hlavní modul obsahující systémové soubory.
 - **Universe**: Modul zprostředkovávající simulátor.
 - **User**: Modul s formuláři a další komponentami pro uživatele.
 - **Utils**: Pomocný modul.
 - **Constants**: Konstanty a konfigurace.
 - **Controllers**: Kontrolery s adresářovou strukturou popisující REST API.
 - **Database**: Databázové soubory.
 - * **Plugins**: Databázové plugíny.
 - * **Schemas**: Databázová schémata.
 - **Global**: Hlavičkové soubory dostupné v globálním prostoru.
 - **Models**: Modelové třídy obsahující aplikační logiku.
 - **Public**: Vstupní HTML soubor a další statické soubory.
 - * **Fonts**: Písma.

- * **Images:** Obrázky rozčleněné do podadresářů.
- * **JavaScript:** Javascriptové soubory.
- **Utils:** Pomocné třídy.

Moduly obsažené zejména v klientské části projektu jsou adresářové celky, které mohou obsahovat další strukturu:

- **Components:** React komponenty.
- **Constants:** Konstanty a konfigurace modulu.
- **Redux:** Akce a reducer.
- **Styles:** Styly.
- **Views:** Pohledy.

Každý modul obsahuje mapování a export všech souborů, které mají být veřejné. Z vnějšku tedy není nutné znát strukturu daného modulu a adresu hledaného souboru uvnitř modulu.

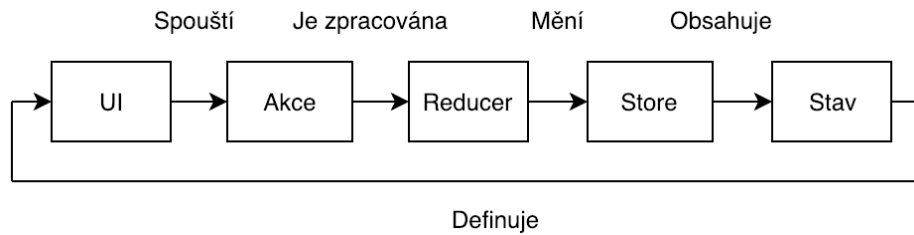
```
import { LoginForm } from '../..//User'
import LoginForm from '../..//User/Components/LoginForm'
```

Zdrojový kód 4: Správné a nesprávné použití importu souboru z modulu

3.2 Uživatelské rozhraní

Klientská část aplikace dodržuje architekturu *Redux*. Všechn stav aplikace je na jednom místě v tzv. *store*, odkud ho mohou číst všechny komponenty, jež jsou na *store* napojené. Kdykoliv se změní stav aplikace, dojde k automatickému překreslení těch částí *UI*, které změněná data obsahují. [6]

Stav aplikace lze změnit zavoláním funkce *dispatch*, jejímž argumentem bude právě prováděná akce. Akce je prostým objektem, který obsahuje povinnou vlastnost **type** s typem akce a libovolné další vlastnosti. Jelikož akce by měly být znovupoužitelné, jsou často umístěny v *action creator*. To je funkce, která akci vrací. [6]



Obrázek 1: Životní cyklus architektury Redux ¹

Samotná změna stavu je pak provedena ve funkci *reducer*. Ten ze starého stavu aplikace a akce vytvoří a vrátí nový stav. Obsahem *reduceru* je často rozsáhlý *switch* s výčtem všech akcí, kde je u každé akce uvedena její modifikace stavu. Aby změna stavu aktualizovala všechny komponenty, které tomuto stavu naslouchají, je nutné, aby *reducer* vracel vždy nově vytvořený objekt, nikoliv pouze pozměněný ten starý. [6]

```

// Actions.ts
export const increment = () => ({ type: ActionTypes.INCREMENT })

// Reducer.ts
case ActionTypes.INCREMENT:
  return { ...state, number: state.number + 1 }

// IncrementButton.ts
class IncrementButton extends React.Component<IProps, IState> {

  public render = (): React.ReactNode => (
    <button onClick={this.props.increment}>
      Increment number {this.props.number}
    </button>
  )
}

export default IncrementButton.connect(
  state => ({ number: state.number }), // mapStateToProps
  { increment } // mapDispatchToProps
)

```

Zdrojový kód 5: Ukázka architektury Redux

¹Vytvořeno v <https://www.draw.io>.

3.2.1 Akce

Vytvářet zejména asynchronní akce manuálně je zdlouhavé. Pro správnou funkcionalitu *UI* je totiž při asynchronním požadavku třeba zaznamenat následující stavy:

- **SENT**: Požadavek byl odeslán,
- **SUCCESS**: Požadavek byl úspěšně ukončen,
- **FAIL**: Požadavek byl neúspěšně ukončen.

V projektu byla proto vytvořena vlastní knihovna *Redux* v modulu *Utils*, která zajišťuje vykonávání několika typů akcí:

- **setAction**: Nastavení hodnoty,
- **toggleAction**: Přepnutí hodnoty (generuje *ON/OFF*),
- **asyncAction**: Asynchronní požadavek (generuje *SENT/SUCCESS/FAIL*).

V kódu pak použití asynchronní akce vypadá následovně. Automaticky je zajištěno sledování průběhu požadavku a je tudíž je jednoduše možné zobrazit např. načítací animaci.

```
export const getBody = (bodyId: string) => (  
  Redux.asyncAction(  
    ActionTypes.GET_BODY,  
    { body: Request.get(Paths.GET_BODY(bodyId)) }  
  )  
)
```

Zdrojový kód 6: Redux akce za využití vlastní knihovny

3.2.2 Reducer

Reducer je běžně funkce, která obsahuje *switch* s výčtem všech akcí a jejich modifikací stavu aplikace. [6] Díky vlastní knihovně to ale není nutné. Stačí uvést pole všech akcí a případně i počáteční stav *store*.

```
export default Redux.createReducer(  
  Object.values(ActionTypes),  
  {  
    isNameVisible: true,  
    areOrbitsVisible: false,  
  })
```

```

        bodies: Redux.EMPTY_ASYNC_ENTITY,
        body: Redux.EMPTY_ASYNC_ENTITY
    }
)

```

Zdrojový kód 7: Redux reducer za využití vlastní knihovny

3.3 3D grafika

Před vykreslováním těles ve vesmíru je nutné ze serveru nejdříve získat data k těmto tělesům. Následovně je použita třída *BodyFactory*, která z datových objektů těles vytvoří kontejnery obsahující kromě samotných dat z databáze také objekty pro vykreslení popsané v podkapitolách této kapitoly.

Celý vesmír je neustále v pohybu. Měsíc obíhá planetu Zemi, která se pohybuje kolem Slunce. Slunce i s celou soustavou obíhá střed naší galaxie. Mléčná dráha se pak volně pohybuje v rámci místní kupy galaxií a ta zas v rámci místní nadkupy galaxií. [1] Počítat absolutní pozici tělesa by proto bylo více než náročné.

Je tedy důležité, aby každé těleso bylo umístěno mezi potomky tělesa, kolem kterého zdánlivě obíhá. Tím je dosaženo dědičnosti a souřadnice není nutné uvádět absolutně vzhledem k vesmíru, ale pouze vzhledem k rodičovskému tělesu. Pro práci s 3D grafikou byla v rámci projektu vytvořena vlastní třída *Scene*. Ta v každém průběhu vykreslovací smyčky vypočítá pozice a rotace těles a aktualizuje je.

```

const scene = new Scene({
    controllable: true,
    element: this.container,
    logarithmicDepth: true,
    objects: this.rootBodies.map(this.bodyFactory.create),
    onRender: this.updateBodies,
    target: 'Slunce'
})

scene.setCameraPosition({ x: 10, y: 20, z: 30 })
scene.setCameraTarget('VY Canis Majoris')

```

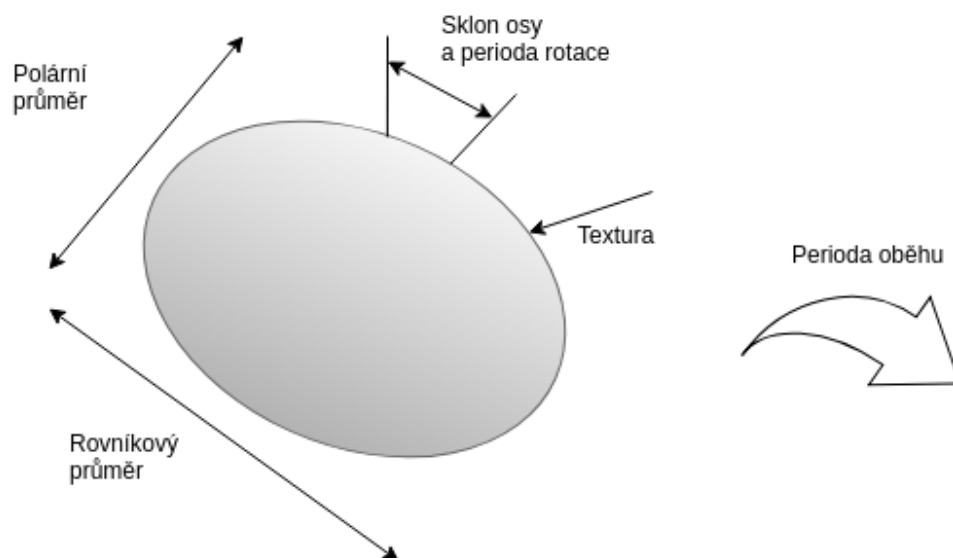
Zdrojový kód 8: Práce s vlastní knihovnou pro 3D grafiku

V každém cyklu dojde u každého z vykreslovaných těles k následujícím krokům:

- **Zjištění počtu milisekund**, které uplynuly od posledního vykreslení,
- **Zjištění vzdálenosti** tělesa od těžiště pomocí knihovní metody v *THREE.js* `bodyPosition.distanceTo(parentBodyPosition)`,
- **Výpočet okamžité rychlosti** dosazením vzdálenosti od těžiště do vzorce představeném v kapitole 3.5.5,
- **Výpočet procentuální části obvodu dráhy**, kterou těleso urazí za jednotku času (dáno dle FPS, v ideálním případě za 16 ms).
- **Nastavení nových souřadnic** tělesa získaných z metody `orbitPath.getPoint(angle)`,
- **Pootočení tělesa kolem osy** o úhel za jednotku času vypočtený z rotační periody,
- **Výpočet vzdáleností** od kamery a od Země,
- **Aktualizace viditelnosti, obsahu a pozice popisku** a orbity tělesa.

3.3.1 Těleso

Samotné těleso je reprezentováno koulí, tedy objektem s geometrií *THREE.SphereGeometry*. Pokud má těleso rozdílný rovníkový a polární průměr, je výsledné zploštění řešeno transformací `new THREE.Matrix4().makeScale()`. Materiálem je *THREE.MeshBasicMaterial* (tělesa emitující světlo) nebo *THREE.MeshPhongMaterial* (tělesa pohlcující světlo). Povrch tělesa tvoří 2D textura *THREE.Texture* tvořená obrázkem ve formátu *JPG*.



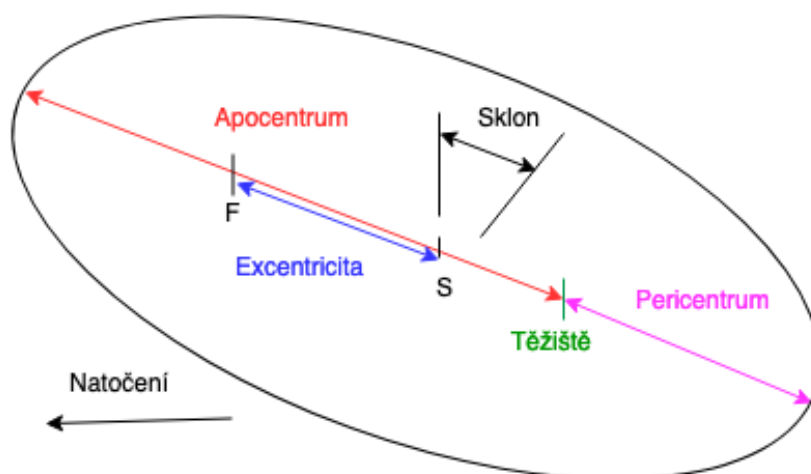
Obrázek 2: Jednoznačná definice tělesa ¹

3.3.2 Orbita

Orbita je objekt s geometrií *THREE.Geometry*, jejíž eliptický tvar je vypočten na základě nejmenší (periapsida) a největší (apoapsida) vzdálenosti tělesa od těžiště a výstřednosti dráhy (excentricita). [1] Jako materiál je použit *THREE.LineMaterial*.

```
const a = (apocenter + pericenter) / 2
const b = Math.sqrt(-Math.pow(a, 2) * eccentricity + Math.pow(a, 2))
const path = new THREE.EllipseCurve(0, 0, a, b, ...)
const geometry = new THREE.Geometry()
geometry.setFromPoints(path.getPoints(ORBIT_SEGMENTS))
```

Zdrojový kód 9: Výpočet geometrie orbity tělesa



Obrázek 3: Jednoznačná definice orbity tělesa ¹

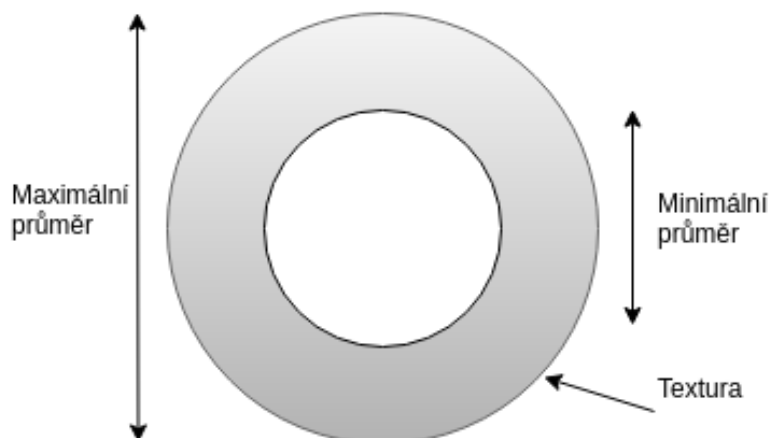
3.3.3 Světlo

Pokud je těleso zdrojem světla, je mezi jeho potomky navíc také bodové světlo *THREE.PointLight*, které emituje světlo o stejné barvě a intenzitě, jakou má reálné těleso.

¹Vytvořeno v <https://www.draw.io>.

3.3.4 Prstence

Prstenec je objekt s geometrií *THREE.BufferRingGeometry*, materiálem *THREE.MeshLambertMaterial* a 2D texturou *THREE.Texture* tvořenou obrázkem ve formátu *JPG* nebo *PNG*. Těleso může mít libovolný počet prstenců, nebo také žádný.



Obrázek 4: Jednoznačná definice prstence tělesa ¹

3.3.5 Popisek

Popisek je reprezentován *HTML* elementem, který je absolutně pozicován na stejné souřadnice, jako je vykreslované těleso. Kromě názvu tělesa obsahuje i aktuální rychlost pohybu, vzdálenost od Země, vzdálenost od kamery a vzdálenost od centrálního tělesa.

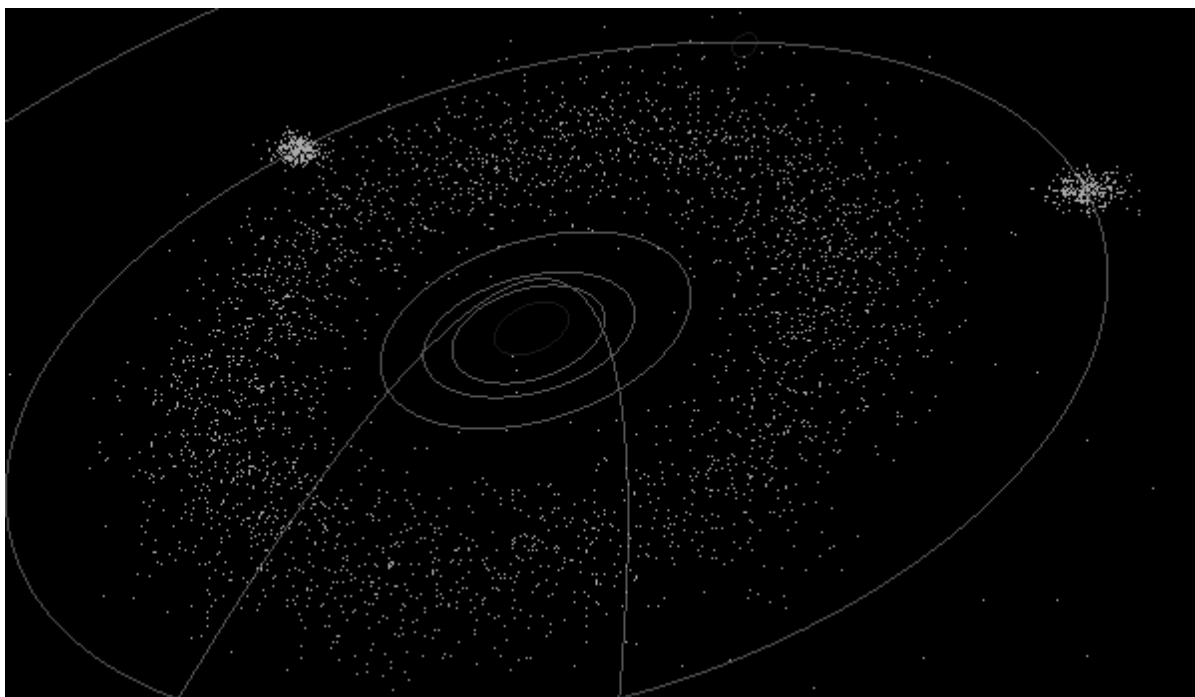
3.3.6 Částice

Protože aplikace obsahuje řádově pouze desítky až stovky těles, je nemožné zobrazit rozsáhlejší struktury, které jsou jinak typické pro danou oblast ve vesmíru. Příkladem může být *Kuiperův pás* nebo *Oortův oblak* v naší sluneční soustavě. Tato uskupení obsahují stovky miliard planetek, komet nebo dalších těles. [1]

Pro tyto účely se v aplikaci využívá náhodné generování částic pomocí *THREE.Points*. V závislosti na typu tělesa (pás, oblak, hvězdokupa, ...) je využito pro rozmístění částic buď rovnoměrné, normální nebo exponenciální rozdělení. Protože zde ale figuruje náhoda,

¹Vytvořeno v <https://www.draw.io>.

nemusí takto vygenerovaná tělesa odpovídat realitě a jedná se pouze o vizuální doplnění prázdného prostoru. Uživatel může zobrazení částic kdykoliv vypnout.



Obrázek 5: Hlavní pás planetek vygenerovaný pomocí částic

Funkce pro generování částic je uložena v databázové kolekci typů těles jako textový řetězec. Je možné tak vytvořit jedinou funkci, např. pro generování spirálních galaxií, a z tohoto typu pak vytvářet instance těles o stejném tvaru, lišících se v ostatních fyzikálních vlastnostech. Protože ale administrátor vše musí nejdříve schválit, je tím zároveň ošetřena bezpečnostní hrozba, kdy by uživatel do generující funkce umístil škodlivý kód.

```
for (let i = 0; i < 1000; i++) {  
  const angle1 = Random.uniform(0, 2 * Math.PI)  
  const angle2 = Random.uniform(0, 2 * Math.PI)  
  
  geometry.vertices.push(new THREE.Vector3(  
    radius * Math.sin(angle1) * Math.cos(angle2),  
    radius * Math.sin(angle1) * Math.sin(angle2),  
    radius * Math.sin(angle1)  
  ))  
}
```

Zdrojový kód 10: Naplnění geometrie částicemi ve tvaru koule

3.3.7 Omezení

Aplikace zanedbává některé z aspektů, které v realitě existují. Důvodem může být vysoká výpočetní náročnost pro prohlížeč nebo složitost implementace. V důsledku toho může v simulátoru docházet k nepřesnostem nebo omezeným možnostem vykreslování těles.

- Hmotnost obíhajících těles je zanedbána. Centrální těleso tedy není ovlivněno tělesy obíhajícími. Oběh je počítán na základě 1. a 2. Keplerova zákona a není třeba řešit vzájemné gravitační ovlivňování těles.
- Tělesa s neperiodickou nebo neeliptickou oběžnou dráhou mají pevně nastavenou pozici a nepohybují se.
- Periapsida se vždy nachází přímo na opačné straně elipsy, než apoapsida. Pokud je oběžná dráha příliš excentrická, může se pozice skutečně nejbližšího bodu k těžišti dráhy a pericentra v aplikaci lišit.
- Jsou zanedbány dlouhodobé události, např. postupné vzdalování Měsíce od Země o 38 mm za rok. [2]

3.4 Serverová část

3.4.1 Zachycení HTTP požadavku

Server vystavuje *REST API* [5], ze kterého si mohou klientské aplikace (v tomto případě pouze webová aplikace) stahovat data. Rozhraní má jasně danou strukturu a jeho implementace a dokumentace je provedena pomocí nástroje *Swagger*. Ten umožňuje sestavit rozhraní z fyzických souborů. Cesta k souboru značí adresu *URI* a obsah zas dostupné metody.

```
// /bodies/{bodyId}.ts
// Routes like [GET] /bodies/10 or [POST] /bodies/20
export default {
  get: (req, res) => res.send('Hello world!')
  put: (req, res) => res.status(401).send('Admin required.')
  post: (req, res) => BodyModel.add(req.body).then(res.send)
  delete: (req, res) => res.send('Body was deleted.')
}
```

Zdrojový kód 11: Definice cesty v REST API

Dokumentace je zpřístupněna na lokální adrese `/api-docs`. Je možné se zde informovat o všech možných *HTTP* požadavcích nebo je s nastavenými parametry simulovat.

3.4.2 Zpracování HTTP požadavku

Jakmile je přijat požadavek od klienta, musí dojít k následujícím krokům:

- Zjištění identity uživatele z tokenu v hlavičce,
- Zkontrolování vyžadovaných oprávnění,
- Obsluha požadavku (načtení a zpracování dat z DB, ...),
- Nastavení HTTP statusu, popř. chybového kódu,
- Odeslání odpovědi.

V aplikaci byla vyvinuta vlastní knihovna, která všechny ze zmíněných procesů automatizuje. Pro většinu druhů zdrojů stačí dva druhy přístupových bodů:

- Pro všechny zdroje,
 - **GET**: Vráť pole zdrojů,
 - **POST**: Vytvoří nový zdroj a vrátí zprávu o úspěchu nebo chybě,
 - **DELETE**: Smaže všechny zdroje a vrátí počet smazaných zdrojů,
- Pro jeden zdroj,
 - **GET**: Vráť jeden zdroj podle ID nebo zprávu o chybě,
 - **PUT**: Upraví jeden zdroj podle ID a vrátí zprávu o úspěchu nebo chybě,
 - **DELETE**: Smaže jeden zdroj podle ID a vrátí zprávu o úspěchu nebo chybě.

Samotná definice přístupových bodů pak může vypadat následovně. U každé přístupové metody je možné určit, jaká oprávnění musí mít uživatel, aby tuto metodu nad daným zdrojem mohl vykonat.

```
// bodies.ts
export default Route.getRouteGroupForAll(
  BodyModel
  { get: Route.all, post: Route.all, delete: Route.onlyAuthenticated }
)
```

```
// bodies/{bodyId}.ts
export default Route.getRouteGroupForOne(
  BodyModel,
  { get: Route.all, put: Route.onlyAdmin, delete: Route.onlyAdmin }
)
```

Zdrojový kód 12: Zpracování HTTP požadavku

Modelové třídy většiny entit není nutné psát individuálně. Všechny mají za úkol poskytnout *CRUD operace* nad danou entitou (např. *BodyModel* nad tělesy), případně generovat notifikace či umožnit práci se schvalovacím procesem. Jediné, co se liší, je název databázové kolekce, vybrané sloupce při výpisu detailu entity a při výpisu všech entit, popř. další parametry. Veškerá logika komunikace s databází byla ukryta do třídy *ItemModel*. Většina dalších modelů je pak pouhou instancí této třídy s konkrétními parametry. *ItemModel* vedle *CRUD* operací automatizuje i vytváření notifikací a schvalovací proces administrátora.

```
// BodyModel.ts
export default new ItemModel<Universis.Universe.Body, Universis.Universe
  .Body.Simple, Universis.Universe.Body.New>(
  dbModel: DatabaseModels.BODY,
  get: {
    selectAll: ['name', 'orbit'],
    joinOne: ['typeId'],
    joinAll: ['typeId']
  },
  add: { approval: true, notification: true }
  update: { approval: true, onAfter: console.log }
  remove: { notification: true, onBefore: onBeforeCallback }
)
```

Zdrojový kód 13: Automatizovaná tvorba modelových tříd

3.5 Databáze

Data v aplikaci se ukládají do databáze *MongoDB* za využití knihovny *Mongoose*. Serverová část se řídí architekturou *MVC* a k databázi mají tedy přístup pouze modelové třídy. Struktura databáze je zobrazena v příloze A na konci tohoto dokumentu.

3.5.1 Připojení do databáze

Připojení do databáze je jednoznačně inicializováno textovým řetězcem obsahujícím jméno a heslo uživatele, adresu serveru, port a název databáze. Dále je třeba zaregistrovat všechna schémata, která budou v aplikaci využita. Pokud kolekce s nově zaregistrovaným schématem neexistuje, dojde k jejímu automatickému vytvoření. Je také možné reagovat na úspěšné připojení nebo na chybu. [5]

Veškerou tuto funkcionalitu obstarává třída *Database*, která byla vytvořena v rámci projektu. Instance této třídy je staticky (tedy pouze jednou, při spuštění serveru [5]) vytvořena ve třídě *Model*, který je předkem všech ostatních modelových tříd. Třída *Model* pak tuto instanci databáze poskytuje jako instanční proměnnou s modifikátorem přístupu `protected` všem svým potomkům.

```
Model.db = new Database({
    username: Config.database.username,
    password: Config.database.password,
    cluster: Config.database.cluster,
    database: Config.database.name,
    onError: console.error,
    schemas: {
        [DatabaseModels.BODY]: BodySchema,
        [DatabaseModels.BODY_EVENT]: BodyEventSchema,
        [DatabaseModels.BODY_TYPE]: BodyTypeSchema,
        [DatabaseModels.TOKEN]: TokenSchema,
        [DatabaseModels.USER]: UserSchema,
        ...
    }
})
```

Zdrojový kód 14: Připojení k databázi v aplikaci

3.5.2 Mongoose schéma

Pomocí schématu lze určit strukturu databázové kolekce a nastavit validační pravidla. Veškerá konfigurace je umístěna v objektu, který se dosadí jako parametr při vytváření nové instance schématu. Je možné také vytvořit např. indexy přes více polí. [5]

```
const UserSchema = new Mongoose.Schema({
```

```

    email: {
      type: String,
      required: [true, 'Email is required.'],
      unique: true,
      validate: { validator: Strings.isEmail }
    },
    ...
  })

```

Zdrojový kód 15: Vytvoření databázového schématu

3.5.3 Mongoose plugin

Pokud je třeba reagovat na jednotlivé události v kolekci (smazání, vložení, ...), lze toho docílit pomocí pluginů. To jsou obdoby spouští z relačních databází. Jedná se o uživatelem definované funkce, jež dostanou v parametru schéma, na kterém jsou zaregistrovány. Na tomto schématu pak mohou definovat operace `pre` nebo `post` s jednotlivými událostmi a jejich obsluhou. [5]

```

// UserSchema.ts
UserSchema.plugin(HashPlugin, { field: 'password' })

// HashPlugin.ts
const HashPlugin = (schema, options) => {
  schema.pre('save', async function () {
    this[options.field] = await Security.hash(this[options.field])
  })
}

```

Zdrojový kód 16: Vytvoření a použití databázového pluginu

Ve zdrojovém kódu výše je zobrazena část pluginu pro automatické hashování hesla. Ten při každé změně daného pole nový obsah zahashuje a teprve onen hash uloží do databáze namísto původní hodnoty. Protože ale není vyloučené, že bude někdy třeba tento plugin použít také v jiném schématu a na jiném poli, plugin byl vytvořen univerzálně. Jméno pole je proto určeno parametrem v pluginu.

Dalším pluginem v aplikaci je *FillBodyPlugin*. Aplikace se snaží o největší možnou automatizaci a je tedy žádoucí, aby se do databáze ukládala pouze nejn nutnější data.

Ostatní data jsou automaticky dopočítána z těch v databázi. Níže je ukázán výpočet únikové rychlosti v_u z tělesa.

$$v_u = \sqrt{\frac{2 * G * m}{r}}$$

G – gravitační konstanta, m – hmotnost tělesa, r – poloměr tělesa [1]

Poněkud složitější je např. vzorec pro výpočet okamžité rychlosti pohybu tělesa po eliptické dráze kolem těžiště. Pro výpočet v různých vzdálenostech od centrálního tělesa je možné měnit pouze hodnotu d .

$$v_{\max} = \frac{\text{const.}}{d} = \frac{2 * S}{T * d} = \frac{2 * \pi * a * b}{T * d} = \frac{2 * \pi * \left(\frac{d_{\max} + d_{\min}}{2}\right)^2 * \sqrt{1 - e^2}}{T * d}$$

S – plocha oběžné dráhy, T – perioda oběhu, d – vzdálenost od tělesa, d_{\max} – apoapsida,

d_{\min} – periapsida, a – velká poloosa, b – malá poloosa, e – excentricita

TODO: Zdroj.

3.5.4 Základní práce s databází

Databáze umožňuje provádět velké množství operací nad daty v ní uloženými. Je možné vyhledávat podle konkrétní hodnoty, pole hodnot, regulárních výrazů nebo číselného rozsahu. Takto vyfiltrovaná data lze číst, smazat či editovat. Je také možné nastavit velké množství parametrů, např. maximální počet dokumentů (*limit*), které vyhovují filtru. Mimo jiné je možné také provádět více dotazů do databáze jako atomické operace pomocí transakcí. [5]

```
const User = this.db.getModel(DatabaseModels.USER)

// User's names, which starts with 'a' and doesn't contain any number.
User.find({ name: /^a[~0-9]*$/ }).select('name').skip(5).limit(10)

// Create new user.
new User({ email: 'email@domail.cz', password: 'heslo' }).save()

// Update user by ID.
User.findByIdAndUpdate(
  '5ba0bef7df0fca0bf99305c1',
  { email: 'newEmail@domain.cz', name: 'newName' }
)
```



```
// Delete all users with names in array, but only if they are adult.  
const names = ['michal', 'username', 'admin']  
User.deleteMany({ name: { $in: names }, age: { $gte: 18 } })
```

Zdrojový kód 17: CRUD operace nad kolekcí uživatelů

3.5.5 Aggregation Framework

V případě složitějších databázových dotazů, na které nestačí předpřipravené metody knihovny *Mongoose*, je možno využít metodu **aggregate**, jež umožňuje pracovat s *Aggregation Frameworkem* databáze *MongoDB*. Ten mimo jiné umožňuje v tzv. *pipeline* dosadit sekvenci dotazů, které se nad daty postupně provedou. Je tak možné používat např. **\$match** (vyhledávání), **\$group** (seskupování) nebo **\$lookup** (spojování více kolekcí). [16]

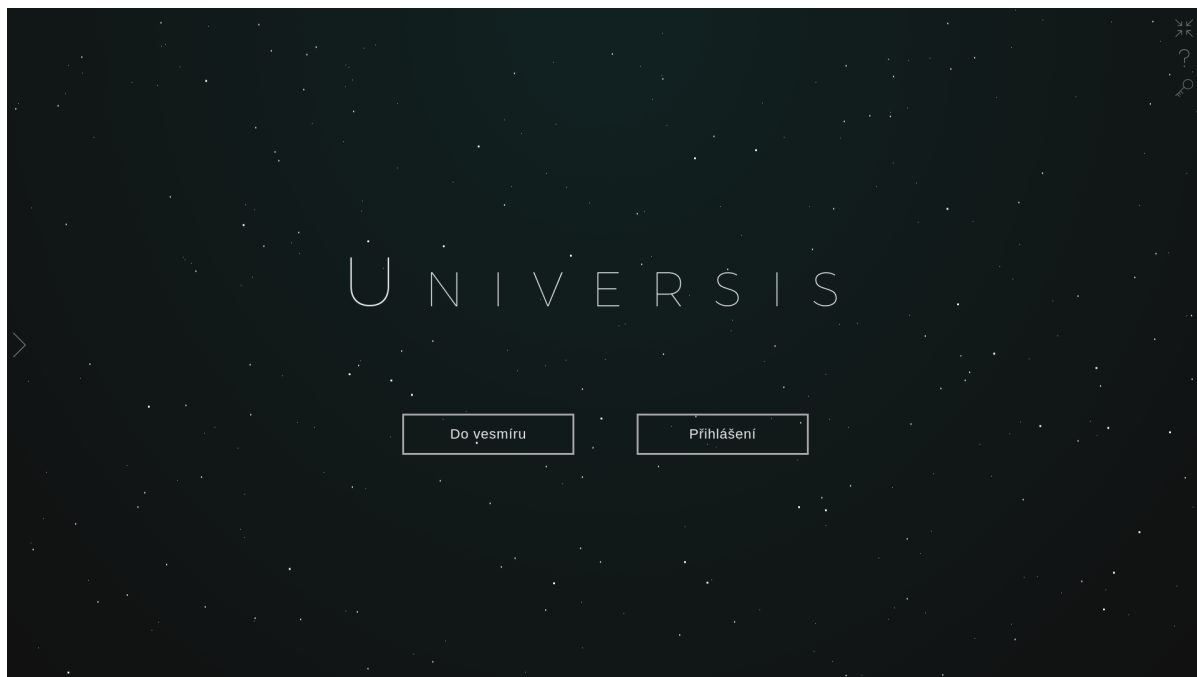
V aplikaci je tento způsob dotazování použit např. v [GET] `/bodies/bodyId`, kdy je třeba z databáze kromě tělesa na základě jeho ID vybrat z jiných kolekcí i typ tělesa, události v historii tělesa, diskuse o tělese, komentáře k diskusím a nakonec i uživatele a hlasy patřící k jednotlivým diskusím a komentářům. Konkrétní podoba dotazu použitého v aplikaci je zobrazena v příloze B na konci tohoto dokumentu.

I přes skutečnost, že *MongoDB* jakožto dokumentová databáze umožňuje zanořování struktur do sebe [10], pro jednodušší práci byly pro tělesa, události, komentáře a hlasy vytvořeny samostatné kolekce.

4 ROZVRŽENÍ APLIKACE

4.1 Hlavní stránka

Hlavní stránka aplikace obsahuje pouze nezbytné odkazy do dalších částí aplikace. Nepřihlášený vidí odkazy do simulátoru a přihlašovací stránku. Přihlášený uživatel vidí taktéž odkaz do simulátoru, ale druhé tlačítko slouží pro odhlášení.



Obrázek 6: Hlavní stránka

4.2 Autentifikace uživatele

4.2.1 Identita

Každý uživatel, jenž se chce přihlásit nebo se zaregistrovat, se dostane na stránku, kde musí prokázat svou identitu zadáním emailu. V závislosti na tom, zda zadaný email již v databázi existuje bude odkázán na stránku pro přihlášení nebo pro registraci.



Obrázek 7: Formulář pro zjištění identity uživatele

4.2.2 Přihlášení

Stránka obsahuje formulář pro zadání hesla. Nachází se zde odkaz zpět a odkaz pro obnovení hesla.



Obrázek 8: Formulář pro přihlášení uživatele

4.2.3 Registrace

Stránka pro zadání a potvrzení hesla k nově vytvářenému účtu. V případě, že uživatel chce změnit registrační email, je možné se vrátit na stránku se zadáváním identity.

The image shows a registration form on a dark background. At the top, the text 'Registrujte se.' is displayed. Below it is a user profile section with a grey silhouette icon and the name 'Michal'. Under the name, there are four colored circles (yellow, grey, grey, orange) each followed by a '0'. Below the profile section, there is a red error message: 'Heslo musí mít 6+ znaků.' followed by two password input fields. The first field has a red underline and three dots indicating a password of length 3. The second field is labeled 'Heslo znovu' and also has three dots. At the bottom, there is a left-pointing arrow and the text 'ZAREGISTROVAT SE' with a right-pointing arrow.

Obrázek 9: Formulář pro registraci uživatele

4.2.4 Zapomenuté heslo

Na této stránce se nachází formulář s polem pro email. Po úspěšném vyplnění se odešle na zadaný email zpráva s odkazem pro obnovení hesla.

4.2.5 Reset hesla

Stránka obsahuje formulář pro nastavení a potvrzení nového hesla. Pro úspěch je nutné, aby se v URL adrese nacházel platný token. Ten se vytvoří v okamžiku vytvoření požadavku na obnovu hesla a má platnost 30 minut.

4.3 Uživatel

4.3.1 Detail uživatele

Na stránce detailu uživatele jsou zobrazeny informace, které o sobě uživatel zveřejnil. Je zde možné tak vidět např. pohlaví, věk, bydliště, profilový text nebo možnosti kontaktu. Mimo to se zde zobrazují i uživatelské statistiky, jež obsahují následující informace:

- Jaká tělesa upravil nebo přidal do databáze,
- Jaké události těles upravil nebo přidal do databáze,

- Kolik diskusí a komentářů má u každého tělesa,
- Kolik kladných a záporných hlasů rozdál,
- Kolik kladných a záporných hlasů obdržel,
- Kolik celkem napsal zpráv.

Výsledkem celkového hodnocení uživatele je jeho reputace. Ta je součtem bodů za:

- **Zlaté medaile** (20) – editorská činnost,
- **Stříbrné medaile** (5) – zakládání diskusí a jejich komentování,
- **Bronzové medaile** (1) – obdržené hlasy a první přihlášení dne.

4.3.2 Editace uživatele

Na této stránce může uživatel změnit některé z informací o něm. Konkrétně se jedná o pohlaví, věk, bydliště, profilový text, veřejný email a adresu webové stránky nebo profilu na některé ze sociálních sítí.

4.4 Simulátor

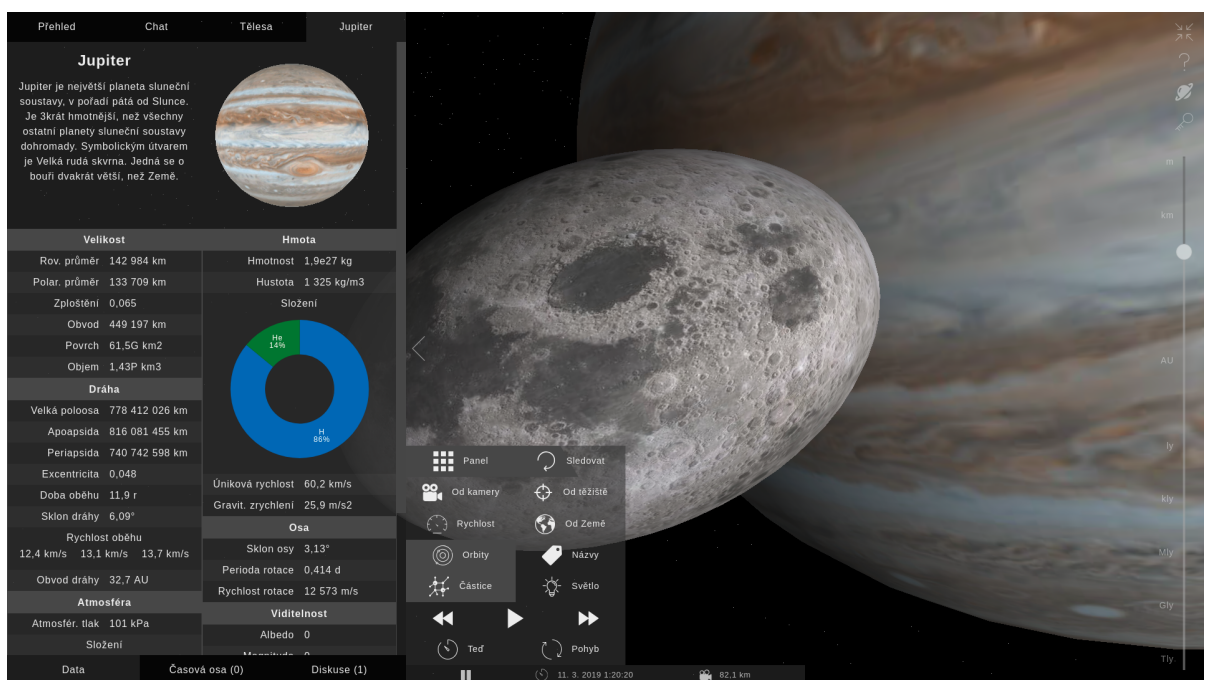
Hlavní náplní aplikace je právě tato stránka. Na $3D$ scéně zobrazuje tělesa ve vesmíru v reálném čase s realistickými poměry velikostí i vzdáleností. Uživatel může myší nebo touchpadem libovolně otáčet kamerou kolem vycentrovaného tělesa.

V pravém dolním rohu se nachází ovládací panel, jenž umožňuje měnit některá nastavení simulátoru. Všechna z nich jsou dostupná i pod klávesovými zkratkami.

- **Panel (P)**: Zobrazí nebo skryje detail právě vycentrovaného tělesa.
- **Sledovat (H)**: Přepíná mezi stálou pozicí kamery, sledováním pohybu tělesa a sledováním pohybu i rotace tělesa.
- **Vzdálenosti od kamery (K)**: Zobrazí nebo skryje vzdálenosti těles od kamery.
- **Vzdálenosti od těžiště (T)**: Zobrazí nebo skryje vzdálenosti těles od těžiště.
- **Vzdálenosti od Země (Z)**: Zobrazí nebo skryje vzdálenosti těles od Země.
- **Rychlost (R)**: Zobrazí nebo skryje okamžité rychlosti těles.
- **Názvy (N)**: Zobrazí nebo skryje názvy těles.
- **Orbity (O)**: Zobrazí nebo skryje orbity těles.

- **Zrychlit (W):** Pokud je čas kladný, zrychlí ho 10krát. Jinak ho 10krát zpomalí.
- **Vrátit rychlost (V):** Nastaví rychlost běhu simulátoru na 1.
- **Zpomalit (S):** Pokud je čas kladný, zpomalí ho 10krát. Jinak ho 10krát zrychlí.
- **Vrátit čas (T):** Nastaví čas simulátoru na aktuální čas.
- **Pohyb (M):** Spustí nebo ukončí rotaci kamery kolem vycentrovaného tělesa.
- **Světlo (L):** Zapne nebo vypne světlo.

Na dolním okraji obrazovky je lišta zobrazující aktuální nastavení a čas simulátoru. Napravo je možné vidět posuvník s aktuální velikostí pohledu.



Obrázek 10: Simulátor

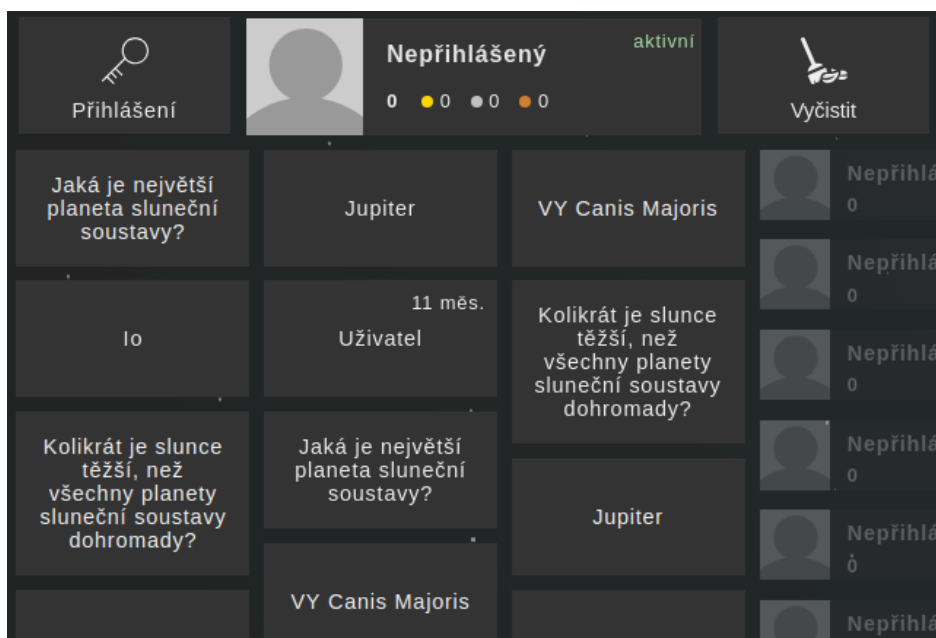
4.5 Uživatelský panel

Uživatelský panel je sekundární okno, ve kterém si uživatel může zobrazovat části aplikace bez nutnosti opouštět aktuální stránku. Zobrazuje se jako poloprůhledný obdélník v levé části obrazovky a obsahuje 4 nezávislé záložky.

4.5.1 Přehled

Výchozí položka v uživatelském panelu je rozdělena na 2 části. Nalevo se zobrazuje posledních 50 událostí na webu a zpráv uživatelů. Je zde i formulář pro napsání nové zprávy.

V případě, že zpráva začíná zavináčem ve tvaru „@uživatel“, bude se jednat o soukromou zprávu pro dále specifikovaného uživatele. Na pravé straně je pak seznam všech uživatelů, které lze řadit buď podle poslední aktivity nebo podle reputace. Přihlášený uživatel zde také vidí stručné informace o svém účtu.



Obrázek 11: Přehled

4.5.2 Seznam těles

Seznam těles umožňuje zobrazit tělesa v databázi i s jejich údaji. Údaje mohou být:

- **Absolutní:** Zobrazí absolutní hodnotu (např. průměr Slunce je 1 392 684 km).
- **Relativní k libovolnému tělesu:** Zobrazí poměr aktuální hodnoty ku hodnotě u porovnávaného tělesa (např. průměr Slunce je roven 109 průměrům Země).

Tělesa lze řadit vzestupně i sestupně podle libovolného kritéria a taktéž je lze podle libovolných kritérií filtrovat za použití následujících vztahů:

- **Obsahuje:** Vyhoví, pokud hodnota obsahuje hledaný text.
- **Je roven:** Vyhoví, pokud je hodnota rovna hledanému textu.
- **Začíná na:** Vyhoví, pokud hodnota začíná hledaným textem.
- **Končí na:** Vyhoví, pokud hodnota končí hledaným textem.
- **Je větší než:** Vyhoví, pokud je hodnota větší, než hledaný text. V případě nečíselné hodnoty vyhoví ty, které se v abecedě nachází později.

- **Je menší než:** Vyhoví, pokud je hodnota menší, než hledaný text. V případě nečíselné hodnoty vyhoví ty, které se v abecedě nachází dříve.

Relativně k Zemi				
Průměr [km]	Je větší než	1013	✕	
Název	Obsahuje			
Název	Průměr	Hmotnost	Hustota	Apocentrum
Země	1	1	1	1
Merkur	0,382	0,055	0,985	0,459
Venuše	0,949	0,815	0,951	0,716
Mars	0,532	0,107	0,714	1,64
Měsíc	0,272	0,012	0,607	0,003

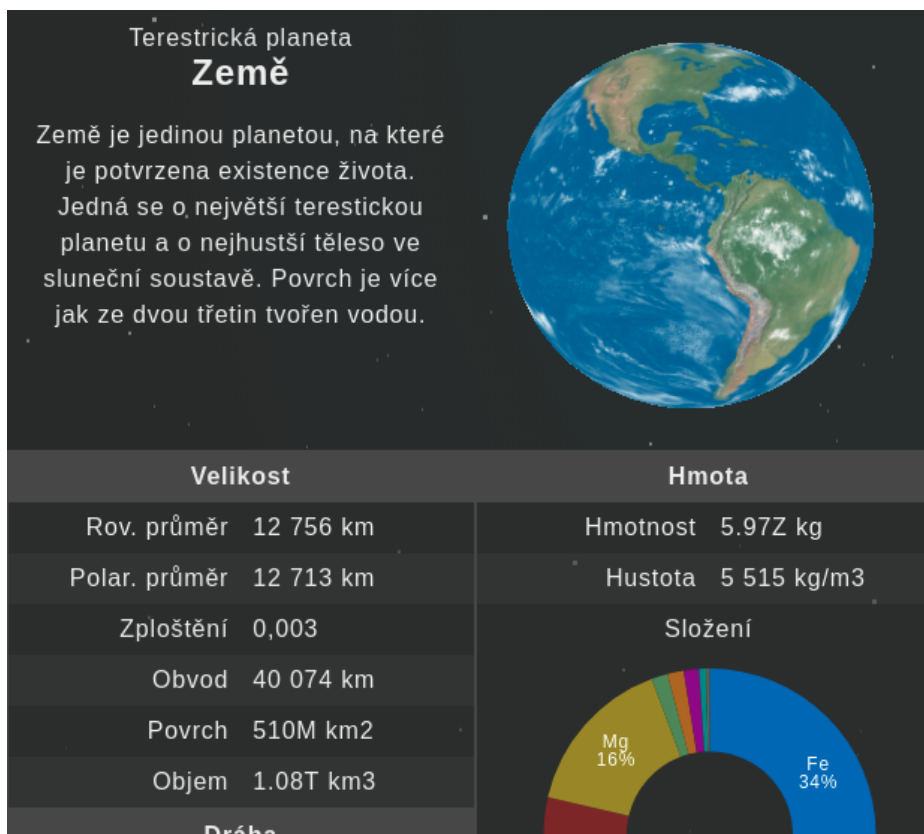
Obrázek 12: Seznam těles

4.5.3 Detail tělesa

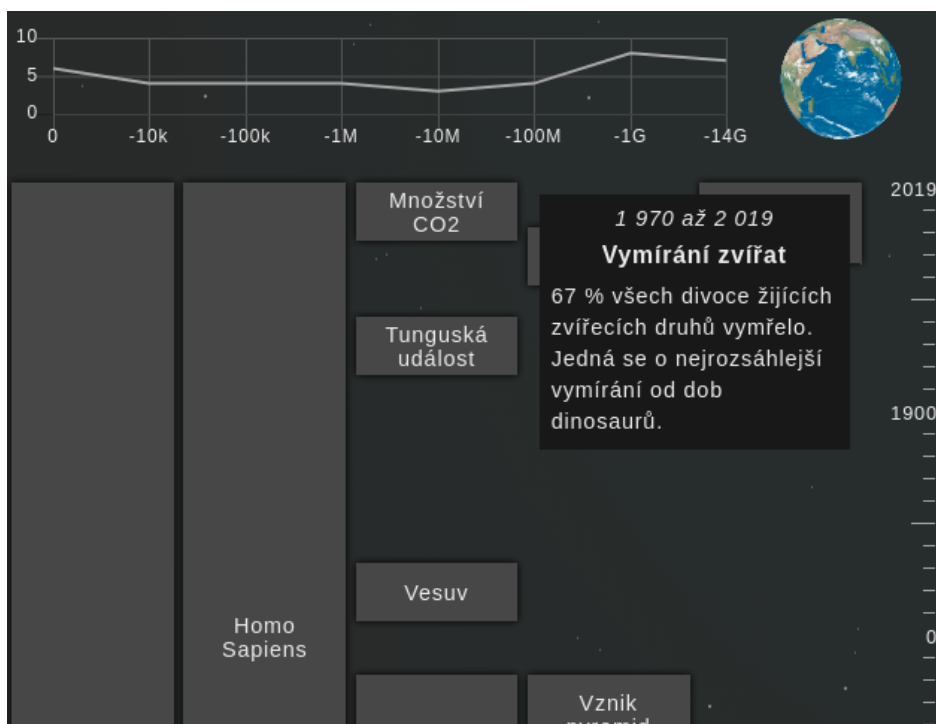
Detail tělesa se skládá ze tří záložek. Výchozí z nich zobrazuje výčet všech známých údajů v tabulce o daném tělese. Součástí je i krátká slovní charakteristika tělesa a jeho 3D vizualizace.

Na druhé záložce se nachází časová osa, na které jsou vyneseny historické události spojené s tělesem. Každá událost obsahuje rok, ve kterém nastala, název a po najetí kurzorem i krátký popis. Je zde i stručný graf zobrazující počet výskytů události v jednotlivých časových obdobích.

Třetí a poslední záložka je určena pro diskuse. Každý zde může reagovat na již existující diskuse, nebo může založit vlastní vlákno. Všechny příspěvky uživatelů lze kladně nebo záporně hodnotit, v důsledku čehož se bude přičítat nebo odečítat reputace autora.





Obrázek 13: Detail tělesa



Obrázek 14: Časová osa tělesa


Diskusí	16	Nejoblíbenější	Václav
Odpovědí	93	Nejaktivnější	Michal
Uživatelů	4	Založit novou diskusi ▾	




 **Jaké jsou podmínky pro vznik života?** Václav 3 d

Život vznikl na této planetě někdy před 4 miliardami let. Jednalo se ale o náhodu, nebo na naší planetě život dříve či později vzniknout musel? Za jakých podmínek může život vzniknout?

👍 2 🗨️ 1 Skrýt odpovědi ^

Vaše odpověď... 

 Olga 2 d

Odpověď na tuto otázku nikdo nezná zcela jistě. Samotný pojem „život“ je dost nejasný. Pokud ale budeme uvažovat život v takové podobě, v jaké ho známe, pak je jednou z podmínek přítomnost vody.

👍 1 🗨️

Obrázek 15: Diskuse o tělese

5 PROBLÉMY ŘEŠENÉ PŘI IMPLEMENTACI

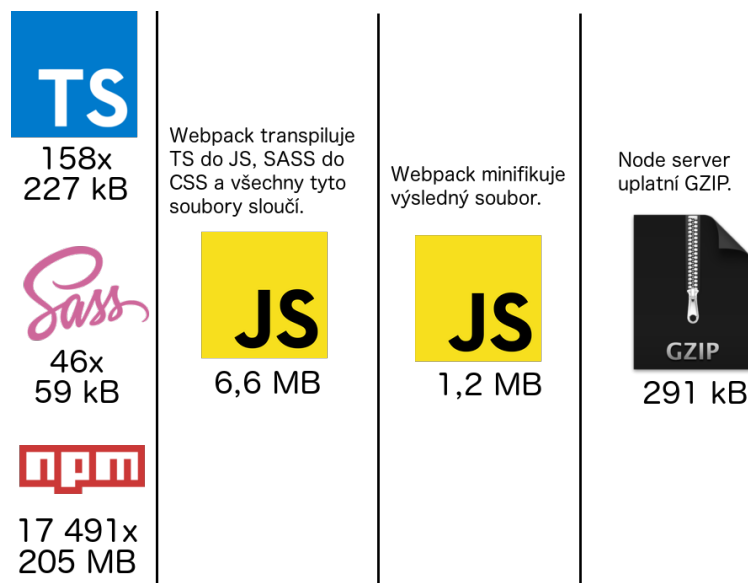
5.1 Omezení viditelnosti těles

V databázi je uloženo velké množství těles. Pokud by se popisky a dráhy všech těles zobrazovaly najednou, bylo by těžké se v simulátoru orientovat. Navíc by tento stav měl negativní důsledky na výkon aplikace. Proto se ve vykreslovací smyčce počítá relativní poloha všech těles vzhledem ke kameře. V závislosti na této poloze se určí, jak se bude dané těleso zobrazovat. Může nastat několik případů, které je nutno ošetřit:

- **Dráha je 40krát větší/menší než obrazovka:** Dráha tělesa bude poloprůhledná, popisky se nebudou zobrazovat.
- **Dráha je 1000krát větší/menší než obrazovka:** Dráha ani popisky se nebudou zobrazovat.
- **Vzdálenost od tělesa je 40krát větší než vzdálenost od centrálního tělesa:** Dráha tělesa bude poloprůhledná, popisky se nebudou zobrazovat.
- **Vzdálenost od tělesa je 1000krát větší než vzdálenost od centrálního tělesa:** Dráha ani popisky se nebudou zobrazovat.

První dva body řeší případy, kdy je těleso příliš oddálené nebo příliš přiblížené. Poslední dva body řeší situaci, ve které má uživatel sice přiměřené přiblížení kamery, nicméně se od tělesa nachází příliš daleko. Příkladem může být situace, kdy si uživatel prohlíží ze vzdálenosti 400 tisíc km prstence planety Saturn. Dráha zemského Měsíce je přiměřeně velká (406 tisíc km), ale přesto by se neměla vykreslovat. Od Saturnu je totiž vzdálena 1,2 miliardy km. [2]

5.2 Sestavení aplikace a optimalizace



Obrázek 16: Sestavení aplikace a optimalizace ^{1, 2, 3}

5.2.1 Transpilace a sloučení JS a CSS souborů

V projektu se nachází velké množství souborů s typescriptovým kódem a souborů se styly. Zahajovat HTTP požadavek pokaždé, kdy si uživatel chce stáhnout jakýkoliv z těchto souborů vyžaduje hodně režije a je to z hlediska času i přenesených dat náročné. Je proto výhodnější všechny tyto soubory sloučit do jediného a ten stáhnout jako jeden celek.

Ze všeho nejdříve je ale nutné transpilovat zdrojové kódy do nativních jazyků. Prohlížeče totiž TypeScriptu ani SASSu nerozumí. TypeScript se transpiluje do JavaScriptu za využití transpilátoru, jenž je dodáván společně s TypeScriptem. Pro převod SASS na CSS slouží Webpack. Ten následně také všechny takto transpilované soubory slouží do jediného javascriptového souboru.

¹Ikona SASS převzata z <https://sass-lang.com/assets/img/logos/logo-b6e1ef6e.svg>

²Ikona NPM převzata z <https://pepa.holla.cz/wp-content/uploads/2016/06/npm.png>

³Ikona GZIP převzata z <https://codeopinion.com/wp-content/uploads/2016/02/gzip.png>

5.2.2 Minifikace a GZIP komprese

I přesto, že jsou všechny zdrojové soubory sloučené do jednoho, stále se jedná o velký objem dat. Je proto vhodné celý soubor pomocí nástroje Webpack minifikovat – odebrat z něj přebytečné mezery, komentáře a zkrátit názvy lokálních proměnných.

Pro ještě větší redukci přenesených dat je dobré nastavit Node.js server tak, aby na všechny odeslané soubory uplatnil GZIP kompresi.

5.3 Bezpečnost a ochrana dat

5.3.1 Autentizace pomocí tokenu

Uživatel se autentizuje při přihlášení svým emailem a heslem, čímž mu je umožněn přístup na jeho účet. Nicméně uživatele je třeba znovu autentizovat i během jeho relace po přihlášení, kdykoliv komunikuje se serverem. Tím lze zabezpečit autorizaci a omezit provádění určitých operací (např. editování tělesa, schvalování úprav, ...) pouze na konkrétní uživatele nebo skupiny uživatelů (např. pouze přihlášení, administrátoři, ...). Nabízí se několik možností v tom, co na server posílat vždy, když je třeba autentifikace:

- **Email a heslo:** Nechat uživatele zadávat přihlašovací údaje pokaždé, když je třeba komunikovat se serverem, by bylo obtěžující. A ukládat heslo na straně klienta není zas bezpečné, protože by zde muselo být v čitelné podobě.
- **ID uživatele:** Jedná se o nebezpečný postup. ID uživatele je veřejný údaj, ke kterému mají přístup i všichni ostatní. Nic by tedy nebránilo útočníkovi poslat *HTTP* požadavek s ID libovolného uživatele.
- **Token:** Ideálním způsobem se zdá být vygenerování pseudonáhodného textového řetězce při přihlášení. Tímto řetězcem se po dobu přihlášení bude uživatel prokazovat. Protože tento řetězec nezná nikdo jiný, než uživatel sám, nikdo jiný se s ním tedy nemůže autentizovat. Navíc proti zneužití má omezenou platnost, po jejímž vypršení je třeba požádat o nový token.

V aplikaci je využit poslední ze zmíněných způsobů. Token má platnost 30 minut. Je tedy zajištěna bezpečná autoritace. Každý může provádět pouze ty operace, na které má skutečně právo.

5.3.2 Hashování hesel

Ukládat hesla v čitelné podobě do databáze není bezpečné. Potenciální útočník, který by prolomil zabezpečení serveru a dostal se k databázi by viděl hesla všech uživatelů. Tento problém řeší *hashování hesel*.

Do databáze se neuloží heslo samotné, ale pouze jeho otisk (*hash*). Když je potřeba porovnat zadané heslo (např. při přihlašování) s heslem, porovnají se jejich otisky. Tím je umožněna autentifikace a zároveň je znemožněno útočníkovi přechytit hesla z databáze.

Pro vyšší bezpečnost se otisk nevypočítává z pouhého hesla, ale hesla spojeného s dalším nijak nesouvisejícím řetězcem. Tím se zabrání možnosti, aby více uživatelů mělo stejné heslo, pokud mají stejná hesla. Tomuto dodatečnému řetězci se říká sůl (*salt*). Často se také otisk vypočítává několikrát za sebou, takže v databázi není uložen otisk hesla, ale otisk, který vznikl na základě jiného otisku.

Hash je výsledkem jednosměrné funkce. To znamená, že z již vypočteného hashe nelze získat původní text. Výjimku tvoří zastaralé *hashovací* funkce, které již byly prolomeny. Je vytvořena databáze pro všechny možné otisky a texty, ze kterých byly vypočítány.

V aplikaci je použita *hashovací funkce bcrypt* s 10 iteracemi a to přímo na úrovni databáze. Celá logika je skryta do tzv. *Mongoose pluginu*. Ten může reagovat na různé události a modifikovat dokument.

```
// UserSchema.ts
UserSchema.plugin(HashPlugin, { field: 'password' })

// HashPlugin.ts
const HashPlugin = (schema, options) => {
  schema.pre('save', async function () {
    this[options.field] = await Security.hash(this[options.field])
  })
}
```

Zdrojový kód 18: Hashování hesel v Mongoose schématu

5.4 Zobrazování hodnot fyzikálních veličin

Aplikace zobrazuje různě velké hodnoty různých fyzikálních veličin. Z důvodu přehlednosti a omezeného množství prostoru, ve kterém se tyto hodnoty zobrazují, byla vytvořena

pomocná třída `Units` umístěná do modulu `Utils`. Ta umožňuje převádět a formátovat jednotky do několika podob.

```
Units.convert(Units.ANGLE.RADIAN, Units.ANGLE.DEGREE, Math.PI) // 180
Units.convert(Units.MASS.KG, Units.MASS.G, 2.5) // 2500
Units.toFull(1234567890, Units.SIZE.KM) // 1 234 567 890 km
Units.toShort(1234567890, Units.SIZE.KM) // 1.23G km
Units.toExponential(1234567890, Units.SIZE.KM) // 1.23e9 km
Units.toFull(123456780, Units.SIZE.KM, Units.SIZE) // 8.25 AU
```

Zdrojový kód 19: Ukázka formátování jednotek

Všechny číselné hodnoty se automaticky zaokrouhlují s rozumnou přesností:

```
Units.toShort(1234.567890) // 1.23k
Units.toShort(123.4567890) // 123
Units.toShort(12.3456790) // 12.3
Units.toShort(1.234567890) // 1.23
Units.toShort(0.0001234567890) // 0.0001
```

Zdrojový kód 20: Ukázka zaokrouhlování hodnot

Není nutné používat předdefinované jednotky v této třídě a lze si vytvořit vlastní. Vytvoření jednotek pro velikost vypadá takto:

```
public static SIZE = {
    M: { value: 1, shortName: 'm' },
    KM: { value: 1000, shortName: 'km' },
    AU: { value: 149597870700, shortName: 'AU' },
    LY: { value: 9461e15, shortName: 'ly' },
    KLY: { value: 9461e18, shortName: 'kly' },
    MLY: { value: 9461e21, shortName: 'Mly' },
    GLY: { value: 9461e24, shortName: 'Gly' },
    TLY: { value: 9461e27, shortName: 'Tly' }
}
```

Zdrojový kód 21: Tvorba vlastních jednotek

5.5 Vykreslování časové osy

Pro vykreslení časové osy představené v podkapitole 4.5.4 *Detail tělesa* byla v rámci projektu vytvořena komponenta *EventsArea*. Ta umožňuje vykreslit pole událostí na 2D ploše v podobě čtyřúhelníků tak, aby si souřadnice jednotlivých událostí odpovídaly. V případě časové osy jsou těmito souřadnicemi počáteční a koncový rok události.

Situace je o to komplikovanější, že časová osa nemusí být lineární a dokonce ani logaritmická. Komponenta proto na vstupu vedle pole událostí přijme i pole hraničních souřadnic. Díky tomu je možné zobrazit roky 2019 až 0 jako jeden díl na časové ose a o něco níže naprosto stejný díl pro roky -1 až -10 000, stejně jako např. -5 mld. až -10 mld.

Překrývání jednotlivých událostí je ošetřeno tak, že se vykreslí do různých sloupců. Je také možno nastavit počet mezistupňů mezi hraničními souřadnicemi. Ty lze nastýlovat např. jako čáry na pravítku pro zlepšení přehlednosti. Při pohybu s kurzorem myši nad oblastí se zobrazuje horizontální čára s aktuální souřadnicí (rokem), nad kterou se kurzor nachází. Pro implementaci komponenty byla použita *CSS* vlastnost *grid*.

```
<EventsArea
  columnsCount={5}
  events={[event1, event2, ...]}
  formatTickValue={Units.toShort}
  minorTicksCount={9}
  tickHeight={15}
  ticks={[new Date().getFullYear(), 0, -1e4, -1e6, -1e8, ...]} />
```

Zdrojový kód 22: Příklad použití komponenty *EventsArea*

ZÁVĚR

POUŽITÁ LITERATURA

- [1] KLECZEK, Josip. Velká encyklopedie vesmíru. Praha: Academia, 2002s [cit. 25. 10. 2018]. ISBN 80-200-0906-x.
- [2] REES, Martin, Vesmír. Přeložil Pavel PŘÍHODA. Praha: Knižní klub, 2006 [cit. 25. 10. 2018]. ISBN 80-242-1668-x.
- [3] ECK, David. *Introduction to Computer Graphics*. [online]. Version 1.2, January 2018. [cit. 22. 2. 2019]. Dostupné z <http://math.hws.edu/eck/cs424/downloads/graphicsbook-linked.pdf>.
- [4] JPL Solar System Dynamics. *JPL Solar System Dynamics* [online]. [cit. 25. 10. 2018] Dostupné z: <https://ssd.jpl.nasa.gov>.
- [5] MARDAN, Azat. Practical Node.js: building real-world scalable web apps. Berkeley, California: Apress, [2014]. Expert's voice in Web development. ISBN 978-1-4302-6595-5.
- [6] BANKS, Alex, PORCELLO, Eve. Learning React: functional web development with React and Redux. Sebastopol, CA: O'Reilly Media, 2017 [cit. 25. 10. 2018]. ISBN: 978-1-4919-5462-1.
- [7] TypeScript. *TypeScript*. [online]. [cit. 22. 2 2019]. Dostupné z: <https://www.typescriptlang.org>.
- [8] Thinking in React. *React*. [online]. [cit. 22. 2. 2019]. Dostupné z: <https://reactjs.org/docs/thinking-in-react.html>.
- [9] MROZEK, Jakub. JavaScript na serveru: Architektura a první Hello World. *Zdroják*. [online]. 5. 10. 2012. [cit. 22. 2. 2019]. ISSN 1803-5620. Dostupné z: <https://www.zdrojak.cz/clanky/javascript-na-serveru-architektura-a-prvni-hello-world/>.
- [10] MROZEK, Jakub. JavaScript na serveru: MongoDB, Mongoose a AngularJS. *Zdroják*. [online]. 26. 10. 2012. [cit. 22. 2. 2019]. ISSN 1803-5620. Dostupné z: <https://www.zdrojak.cz/clanky/javascript-na-serveru-mongodb-mongoose-angularjs>.

- [11] Úvod - Základy systému Git. *git*. [online]. [cit. 22. 2. 2019]. Dostupné z: <https://git-scm.com/book/cs/v1/%C3%A9vod-Z%C3%A1klady-syst%C3%A9mu-Git>.
- [12] Concepts. *webpack*. [online]. [cit. 22. 2. 2019]. Dostupné z: <https://webpack.js.org/concepts>.
- [13] Fantastic front-end performance. *HACKS*. [online]. 20 2 2019. [cit. 22. 2. 2019]. Dostupné z: <https://hacks.mozilla.org/2012/12/fantastic-front-end-performance-part-1-concatenate-compress-cache-a-node-js-holiday-season-part-4/>.
- [14] Sass Basics. *Sass*. [online]. [cit. 22. 2. 2019]. Dostupné z: <https://sass-lang.com/guide>.
- [15] File Structure. *React*. [online]. [cit. 22. 2. 2019]. Dostupné z: <https://reactjs.org/docs/faq-structure.html>.
- [16] Aggregate. *mongoose*. [online]. [cit. 22. 2. 2019]. Dostupné z: <https://mongoosejs.com/docs/api.html#Aggregate>.
- [17] JOHNSON, Tom. Swagger UI tutorial. *Documenting APIs*.. [online]. c2019 [cit. 22. 2. 2019]. Dostupné z: https://idratherbewriting.com/learnapidoc/pubapis_swagger.html.
- [18] KAVIČKA, Antonín. Pardubice. Elektronické sylaby přednášek předmětu Modelování a simulace. [cit. 22. 2. 2019].

SEZNAM PŘÍLOH

Příloha A – Struktura databáze	53
Příloha B – Využití aggregation frameworku	??
Příloha C – Externí balíčky NPM použité v aplikaci	??

PŘÍLOHA A – STRUKTURA DATABÁZE

PŘÍLOHA B – VYUŽITÍ AGGREGATION FRAMEWORKU

```
this.db.getModel(DatabaseModels.BODY).aggregate([
  { $match: { _id: ObjectId('507f1f77bcf86cd799439011') } },
  { $lookup: {
    from: 'bodytypes',
    localField: 'typeId',
    foreignField: '_id',
    as: 'type'
  } },
  { $lookup: {
    from: 'bodyevents',
    localField: '_id',
    foreignField: 'bodyId',
    as: 'events'
  } },
  { $lookup: {
    from: 'bodyposts',
    let: { 'bodyId': '$_id' },
    pipeline: [
      { $match: { $expr: { $eq: ['$bodyId', '$$bodyId'] } } },
      { $lookup: {
        from: 'bodyposts',
        let: { 'discussionId': '$_id' },
        pipeline: [
          { $match: {
            $expr: { $eq: ['$discussionId', '$$discussionId'] } }
          },
          { $lookup: {
            from: 'postvotes',
            let: { 'postId': '$_id' },
            pipeline: [
              { $match: {
                $expr: { $eq: ['$postId', '$$postId'] } }
              }
            ]
          }
        ]
      }
    ]
  }
}
```

```

        as: 'votes'
      } }
    ],
    as: 'answers'
  } },
  { $lookup: {
    from: 'postvotes',
    let: { 'postId': '$_id' },
    pipeline: [
      { $match: { $expr: { $eq: ['$postId', '$$postId'] } } }
    ],
    as: 'votes'
  } }
],
as: 'discussions'
} },
{ $project: { typeId: 0 } }
])

```

PŘÍLOHA C – EXTERNÍ BALÍČKY NPM

POUŽITÉ V APLIKACI

Backend

bcrypt ^2.0.1
body-parser ^1.18.3
compression ^1.7.2
express ^4.16.3 (@types/express ^4.16.1)
express-openapi ^1.10.0
http-status-codes ^1.3.0
jsonwebtoken ^8.3.0 (@types/jsonwebtoken ^8.3.2)
mongoose ^5.4.11 (@types/mongoose ^5.3.21)
path ^0.12.7
swagger-ui-express ^3.0.10
nodemon ^1.17.15
npm-run-all ^4.1.3

Frontend

axios ^0.18.0
chart.js ^2.7.3
chartjs-plugin-labels ^1.1.0
classnames ^2.2.6
js-cookie ^2.2.0 (@types/js-cookie ^2.2.0)
node-sass ^4.9.3
path ^0.12.7
react ^16.5.2 (@types/react ^16.8.7)
react-chartjs-2 ^2.7.4
react-dom ^16.5.2 (@types/react-dom ^16.0.8)
react-masonry-component ^6.2.1
react-rangeslider ^2.2.0
react-redux ^5.0.7 (@types/react-redux ^5.0.21)

react-router-dom ^4.3.1 (@types/react-router-dom ^4.3.1)
redux-form ^7.4.2 (@types/redux-form ^7.4.8)
redux-thunk ^2.2.0
screenfull ^3.3.3
three ^0.94.0 (@types/three ^0.92.24)
three-trackballcontrols ^0.0.7
css-loader ^0.28.9
postcss-loader ^2.1.6
sass-loader ^6.0.6
style-loader ^0.20.2
ts-loader ^5.2.1
typescript ^3.1.1
webpack ^4.20.2
webpack-cli ^3.1.2
webpack-dev-server ^3.1.9 (@types/webpack-dev-server ^2.9.6)

PŘÍLOHA D – SEZNAM SOUBORŮ ZDROJOVÝCH KÓDŮ NA PŘILOŽENÉM NOSIČI

.gitignore
package-lock.json
package.json
src/Client/package-lock.json
src/Client/package.json
src/Client/postcss.config.js
src/Client/src/Charts/Components/DonutChart.tsx
src/Client/src/Charts/Components/LineChart.tsx
src/Client/src/Charts/index.ts
src/Client/src/Controls/Components/AboutControl.tsx
src/Client/src/Controls/Components/Alert.tsx
src/Client/src/Controls/Components/AuthenticationControl.tsx
src/Client/src/Controls/Components/ContextInfo.tsx
src/Client/src/Controls/Components/ContextMenu.tsx
src/Client/src/Controls/Components/ContextTrigger.tsx
src/Client/src/Controls/Components/Control.tsx
src/Client/src/Controls/Components/ControlLink.tsx
src/Client/src/Controls/Components/ControlPanel.tsx
src/Client/src/Controls/Components/FullScreenControl.tsx
src/Client/src/Controls/Components/HelpControl.tsx
src/Client/src/Controls/Components/HomeControl.tsx
src/Client/src/Controls/Components/UIControl.tsx
src/Client/src/Controls/Components/ViewSizeControl.tsx
src/Client/src/Controls/Styles/Alert.scss
src/Client/src/Controls/Styles/Context.scss
src/Client/src/Controls/Styles/ContextInfo.scss
src/Client/src/Controls/Styles/ControlPanel.scss
src/Client/src/Controls/Styles/Controls.scss
src/Client/src/Controls/index.scss

src/Client/src/Controls/index.ts
src/Client/src/Forms/Components/Back.tsx
src/Client/src/Forms/Components/Field.tsx
src/Client/src/Forms/Components/FlexRow.tsx
src/Client/src/Forms/Components/Form.tsx
src/Client/src/Forms/Components/Note.tsx
src/Client/src/Forms/Components/Select.tsx
src/Client/src/Forms/Components/Submit.tsx
src/Client/src/Forms/Components/Title.tsx
src/Client/src/Forms/index.scss
src/Client/src/Forms/index.ts
src/Client/src/Global/IBodyContrainer.ts
src/Client/src/Global/IFilter.ts
src/Client/src/Global/ILinkButton.ts
src/Client/src/Global/IScene.ts
src/Client/src/Global/IStrings.ts
src/Client/src/Global/IUniverse.ts
src/Client/src/Global/IUserScore.ts
src/Client/src/Global/Redux.ts
src/Client/src/Global/Select.ts
src/Client/src/Global/Table.ts
src/Client/src/Global/Units.ts
src/Client/src/Panel/Components/AnswerForm.tsx
src/Client/src/Panel/Components/Bodies.tsx
src/Client/src/Panel/Components/BodiesFilterForm.tsx
src/Client/src/Panel/Components/BodiesSettingsForm.tsx
src/Client/src/Panel/Components/Body.tsx
src/Client/src/Panel/Components/BodyData.tsx
src/Client/src/Panel/Components/BodyDiscussion.tsx
src/Client/src/Panel/Components/BodyPost.tsx
src/Client/src/Panel/Components/BodyTimeline.tsx
src/Client/src/Panel/Components/Chat.tsx

src/Client/src/Panel/Components/DiscussionForm.tsx
src/Client/src/Panel/Components/Notifications.tsx
src/Client/src/Panel/Components/Overview.tsx
src/Client/src/Panel/Components/Panel.tsx
src/Client/src/Panel/Components/People.tsx
src/Client/src/Panel/Redux/ActionTypes.ts
src/Client/src/Panel/Redux/PanelActions.ts
src/Client/src/Panel/Redux/PanelReducer.ts
src/Client/src/Panel/Styles/Bodies.scss
src/Client/src/Panel/Styles/Body.scss
src/Client/src/Panel/Styles/Chat.scss
src/Client/src/Panel/Styles/Notifications.scss
src/Client/src/Panel/Styles/Overview.scss
src/Client/src/Panel/Styles/Panel.scss
src/Client/src/Panel/Styles/People.scss
src/Client/src/Panel/index.scss
src/Client/src/Panel/index.tsx
src/Client/src/System/Components/AnimatedBackground.tsx
src/Client/src/System/Components/App.tsx
src/Client/src/System/Constants/Strings.ts
src/Client/src/System/Redux/ActionTypes.ts
src/Client/src/System/Redux/Store.ts
src/Client/src/System/Redux/SystemActions.ts
src/Client/src/System/Redux/SystemReducer.ts
src/Client/src/System/Styles/AnimatedBackground.scss
src/Client/src/System/Styles/Home.scss
src/Client/src/System/Views/HomeView.tsx
src/Client/src/System/index.scss
src/Client/src/System/index.ts
src/Client/src/Universe/Components/BodyPreview.tsx
src/Client/src/Universe/Components/Canvas.tsx
src/Client/src/Universe/Components/ControlBar.tsx

src/Client/src/Universe/Components/ControlPanel.tsx
src/Client/src/Universe/Components/UI.tsx
src/Client/src/Universe/Constants/Config.ts
src/Client/src/Universe/Constants/Keys.ts
src/Client/src/Universe/Constants/Visibility.ts
src/Client/src/Universe/Redux/ActionTypes.ts
src/Client/src/Universe/Redux/UniverseActions.ts
src/Client/src/Universe/Redux/UniverseReducer.ts
src/Client/src/Universe/Styles/ControlBar.scss
src/Client/src/Universe/Styles/Controls.scss
src/Client/src/Universe/Styles/UI.scss
src/Client/src/Universe/Utils/BodyContainer.ts
src/Client/src/Universe/Utils/BodyFactory.ts
src/Client/src/Universe/Utils/Listener.ts
src/Client/src/Universe/Utils/Scene.ts
src/Client/src/Universe/Utils/TextureStore.ts
src/Client/src/Universe/Utils/Universe.ts
src/Client/src/Universe/Views/UniverseView.tsx
src/Client/src/Universe/index.scss
src/Client/src/Universe/index.ts
src/Client/src/User/Components/IdentityForm.tsx
src/Client/src/User/Components/LoginForm.tsx
src/Client/src/User/Components/SignUpForm.tsx
src/Client/src/User/Components/UserInfo.tsx
src/Client/src/User/Components/UsersList.tsx
src/Client/src/User/Redux/ActionTypes.ts
src/Client/src/User/Redux/UserActions.ts
src/Client/src/User/Redux/UserReducer.ts
src/Client/src/User/Styles/UserInfo.scss
src/Client/src/User/Views/IdentityView.tsx
src/Client/src/User/Views/LoginView.tsx
src/Client/src/User/Views/SignUpView.tsx

src/Client/src/User/index.scss
src/Client/src/User/index.ts
src/Client/src/Utils/Components/AsyncEntity.tsx
src/Client/src/Utils/Components/BlurLayout.tsx
src/Client/src/Utils/Components/Component.tsx
src/Client/src/Utils/Components/DataTable.tsx
src/Client/src/Utils/Components/Dropdown.tsx
src/Client/src/Utils/Components/EventsArea.tsx
src/Client/src/Utils/Components/FadeLayout.tsx
src/Client/src/Utils/Components/Link.tsx
src/Client/src/Utils/Components/Loader.tsx
src/Client/src/Utils/Components/Menu.tsx
src/Client/src/Utils/Components/QueryMenu.tsx
src/Client/src/Utils/Components/RelativeTime.tsx
src/Client/src/Utils/Components/SimpleComponent.tsx
src/Client/src/Utils/Components/StatelessComponent.tsx
src/Client/src/Utils/Components/Table.tsx
src/Client/src/Utils/Components/ToggleLayout.tsx
src/Client/src/Utils/Components/UUILayout.tsx
src/Client/src/Utils/Components/View.tsx
src/Client/src/Utils/Constants/Config.ts
src/Client/src/Utils/Constants/CookieExpirations.ts
src/Client/src/Utils/Constants/Cookies.ts
src/Client/src/Utils/Constants/Queries.ts
src/Client/src/Utils/Constants/Titles.ts
src/Client/src/Utils/Constants/Urls.ts
src/Client/src/Utils/Styles/BlurLayout.scss
src/Client/src/Utils/Styles/Constants.scss
src/Client/src/Utils/Styles/DataTable.scss
src/Client/src/Utils/Styles/Default.scss
src/Client/src/Utils/Styles/Dropdown.scss
src/Client/src/Utils/Styles/EventsArea.scss

src/Client/src/Utils/Styles/Loader.scss
src/Client/src/Utils/Styles/Table.scss
src/Client/src/Utils/Styles/ToggleLayout.scss
src/Client/src/Utils/Styles/Utils.scss
src/Client/src/Utils/Utils/Cookies.ts
src/Client/src/Utils/Utils/Dates.ts
src/Client/src/Utils/Utils/Filter.ts
src/Client/src/Utils/Utils/Html.ts
src/Client/src/Utils/Utils/Redux.ts
src/Client/src/Utils/Utils/Request.ts
src/Client/src/Utils/Utils/Units.ts
src/Client/src/Utils/Utils/Url.ts
src/Client/src/Utils/Utils/Words.tsx
src/Client/src/Utils/index.scss
src/Client/src/Utils/index.ts
src/Client/src/index.scss
src/Client/src/index.tsx
src/Client/webpack.config.js
src/Constants/Config.ts
src/Constants/DatabaseModels.ts
src/Constants/Errors.ts
src/Constants/NotificationSubjects.ts
src/Constants/Operations.ts
src/Constants/index.ts
src/Controllers/bodies.ts
src/Controllers/bodies/count.ts
src/Controllers/bodies/events/eventId.ts
src/Controllers/bodies/bodyId.ts
src/Controllers/bodies/bodyId/count.ts
src/Controllers/bodies/bodyId/events.ts
src/Controllers/bodies/bodyId/posts.ts
src/Controllers/bodyTypes.ts

src/Controllers/bodyTypes/count.ts
src/Controllers/bodyTypes/bodyTypeId.ts
src/Controllers/login.ts
src/Controllers/notifications.ts
src/Controllers/posts/votes/voteId.ts
src/Controllers/posts/postId/posts.ts
src/Controllers/posts/postId/votes.ts
src/Controllers/users.ts
src/Controllers/users/count.ts
src/Controllers/users/userId.ts
src/Database/Database.ts
src/Database/DatabaseModel.ts
src/Database/Plugins/FillBodyPlugin.ts
src/Database/Plugins/HashPlugin.ts
src/Database/Plugins/ModifyPlugin.ts
src/Database/Schemas/BodyEventSchema.ts
src/Database/Schemas/BodyPostSchema.ts
src/Database/Schemas/BodySchema.ts
src/Database/Schemas/BodyTypeSchema.ts
src/Database/Schemas/NotificationSchema.ts
src/Database/Schemas/PostVoteSchema.ts
src/Database/Schemas/TokenSchema.ts
src/Database/Schemas/UnapprovedSchema.ts
src/Database/Schemas/UserSchema.ts
src/Global/Database.ts
src/Global/Discussion.ts
src/Global/Error.ts
src/Global/Event.ts
src/Global/FunctionalInterfaces.ts
src/Global/Functions.ts
src/Global/IBody.ts
src/Global/IBodyType.ts

src/Global/IFactory.ts
src/Global/IObject.ts
src/Global/ISecurityModel.ts
src/Global/IServer.ts
src/Global/IUser.ts
src/Global/Item.ts
src/Global/Notification.ts
src/Global/Routes.ts
src/Global/Structures.ts
src/Global/Universe.ts
src/Global/User.tsx
src/Models/BodyEventModel.ts
src/Models/BodyModel.ts
src/Models/BodyPostModel.ts
src/Models/BodyTypeModel.ts
src/Models/ItemModel.ts
src/Models/Model.ts
src/Models/NotificationModel.ts
src/Models/PostVoteModel.ts
src/Models/SecurityModel.ts
src/Models/UserModel.ts
src/Public/index.html
src/Server.ts
src/Swagger.ts
src/Utils/Arrays.ts
src/Utils/Dates.ts
src/Utils/Physics.ts
src/Utils/Route.ts
src/Utils/Security.ts
src/Utils/Strings.ts
src/index.ts

tsconfig.json