

Gymnázium Arabská, Praha 6, Arabská 14

Programování

ROČNÍKOVÁ PRÁCE



Gymnázium Arabská, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

ROČNÍKOVÁ PRÁCE

Předmět: Programování

Téma: Biocross

Autor: Kryštof Maxa

Třída: 4.E

Školní rok: 2024/2025

Vedoucí práce: Mgr. Jan Lána

Třídní učitel: Mgr. Blanka Hniličková

Prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona č. 121/2000 Sb. (tzv. autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne 30. 3. 2025

Kryštof Maxa

Poděkování

Rád bych poděkoval panu Mgr. Šimonu Hrozinkovi za konzultace, během nichž jsme diskutovali, co by mělo být uvedeno v materiálu pro výuku Mendelových zákonů a jak by mělo správně probíhat křížení na stránce Crossing.

Anotace

Biocross je webová aplikace sloužící jako učební pomůcka pro výuku Mendelových zákonů. Téma Mendelovy zákony bylo pro mě i mé spolužáky obtížné na pochopení, což mě inspirovalo k vytvoření této aplikace. Jejím cílem je poskytnout studentům přehledné a vizuálně zpracované vysvětlení Mendelových zákonů, doplněné o ilustrativní obrázky, a následně umožnit praktické procvičení genetického křížení. Biocross obsahuje čtyři hlavní části: "Domovská stránka", "Pojmy", "Crossing" a "Classroom". Domovská stránka poskytuje přehled Mendelových zákonů, stránka Pojmy vysvětluje odborné pojmy, stránka Crossing umožňuje simulaci genetických křížení a Classroom slouží k řešení úkolů zadaných vyučujícím.

Klíčová slova

Webová aplikace, Mendelovy zákony, Node.js, Vzdělávání, Učební pomůcka

Annotation

Biocross is a web application designed as a learning tool for teaching Mendel's laws. The topic of Mendel's laws was challenging for both me and my classmates to understand, which inspired me to create this application. Its goal is to provide students with a clear and visually processed explanation of Mendel's laws, supplemented with illustrative images, and subsequently enable practical exercises in genetic crossing. Biocross consists of four main sections: "Home Page," "Concepts," "Crossing," and "Classroom." The Home Page provides an overview of Mendel's laws, the Concepts section explains technical terms, the Crossing section allows for genetic crossing simulations, and the Classroom section is used for solving assignments given by the teacher.

Keywords

Web application, Mendel's laws, Node.js, Education, Learning tool

Zadání ročníkového projektu

Moje maturitní práce slouží jako učební pomůcka pro jednodušší pochopení tématu mendelových zákonů. Jde o webovou stránku, kvůli jednoduchému přístupu pro studenty. Jednoduššího pochopení docílíme vizualizací — např. barevně, procenta, obrázky...

Webovou stránku bude využívat také učitel, který má možnost vytvořit učebnu, kde budou mít studenti přístup k domácím úkolům.

Obsah

1	Úvod	1
2	Mendelovy zákony	2
2.1	Gregor Johann Mendel	2
2.1.1	Něco o životě	2
2.1.2	Objevení Mendelových zákonů	2
2.2	Základní pojmy genetiky	2
2.3	1. Mendelův zákon	3
2.4	2. Mendelův zákon	3
2.5	3. Mendelův zákon – Zákon o nezávislé kombinovatelnosti alel	3
2.5.1	Úplná dominance	4
2.5.2	Neúplná dominance	4
2.5.3	Kodominance	4
3	Funkce aplikace Biocross	5
3.1	Domovská stránka	5
3.1.1	Zvolení designu	5
3.1.2	Složení flashcards	5
3.2	Pojmy	7
3.3	Crossing	8
3.3.1	Co je to Crossing	8
3.3.2	Jak na Crossing	8
3.3.3	Více variant jak měl Crossing vypadat	10
3.4	Classroom	11

3.4.1	Použití jako student	11
3.4.2	Použití jako učitel	12
4	Design aplikace	13
4.1	Architektura	13
4.2	Prezentační vrstva (Frontend)	14
4.2.1	Principy	14
4.2.2	Crossing a křížení	17
4.3	Vrstva aplikační logiky (Backend)	20
4.3.1	Vytvoření kostry a základního nastavení aplikace	20
4.3.2	Principy	22
4.3.3	Seznam Endpointů	25
4.4	Datová vrstva (Database)	26
4.4.1	Použitá databáze	26
4.4.2	Schéma databáze	26
4.4.3	Instalace databázové knihovny	27
4.4.4	Připojení k databázi	27
4.4.5	Příklad použití	28
5	Deployment	29
5.1	Deployment do hostovacích center	29
5.2	Instalace v lokálním prostředí	29
5.2.1	Požadavky	29
5.2.2	Stážení aplikace	30
5.2.3	Instalace závislostí	30
5.2.4	Spuštění aplikace	30

6 Závěr	31
6.1 Budoucí rozvoj	31
7 Speciální citace	32
Seznam obrázků	33
Seznam ukázek kódu	33

1 Úvod

Účel mé aplikace spočívá v usnadnění pochopení **Mendelových zákonů**, jejich osvojení a následném procvičení. Z vlastní zkušenosti vím, že toto téma je velmi náročné na porozumění. Nejen já, ale i moji spolužáci jsme měli problém **Mendelovy zákony** správně pochopit a naučit se je. Právě tato zkušenost mě inspirovala k vytvoření této práce. Cílem aplikace je pomoci ostatním studentům, neboť vhodná vizualizace a názorné procvičení by měly přispět k lepšímu pochopení této problematiky.

V první části mé maturitní práce se věnuji samotným Mendelovým zákonům. Následně popisuji, jak moji aplikaci používat, a poté se zaměřuji na její architekturu, kde detailněji rozebírám její strukturu, použité technologie a způsob implementace.

Pro vývoj aplikace jsem použil technologie **HTML**, **CSS**, **JavaScript**, **SQL**, **Node.js** a **EJS**. Uživatelské rozhraní aplikace jsem vytvořil pomocí **EJS**, který umožňuje dynamické generování **HTML** šablon. **CSS** jsem využil pro vizuální úpravu a **JavaScript** pro interaktivní prvky webové stránky. Serverovou část aplikace jsem implementoval v **Node.js**, který využívá jazyk **JavaScript**, a použil jsem framework **Express**.

Pro práci s databází jsem použil **MySQL**. Aplikace je nasazena pomocí služby **Render.com** a databáze je hostována online na platformě **Aiven.com**. Jako vývojové prostředí jsem použil **Visual Studio Code**.

2 Mendelovy zákony

V této kapitole se budu snažit přiblížit téma Mendelovy zákony, jelikož je to stavební kámen pro pochopení toho, co Biocross dokáže. Postupně vysvětlím jednotlivé zákony pomocí odborné literatury a konkrétních příkladů.

2.1 Gregor Johann Mendel

2.1.1 Něco o životě

Prvně bych uvedl pár vět k panu G. J. Mendelovi, jenž je autorem Mendelových zákonů. G. J. Mendel se narodil 20. července 1822 do německy mluvící rodiny v Hynčicích (vesnice na území dnešního Slezska). Z vesnické školy přešel na gymnázium v Opavě. V roce 1843 vstoupil Johann Mendel do augustiniánského řádu v klášteře na Starém Brně. Podařilo se mu dostat na studia do Vídně, která byla klíčová pro pozdější pokusy s hrachem setým a dalšími rostlinami.

2.1.2 Objevení Mendelových zákonů

Gregor Mendel zkoumal dědičnost pomocí křížení hrachu a zjistil, že určité znaky se předávají podle pravidelných poměrů. Pečlivě analyzoval výsledky a formuloval tři zákony dědičnosti. *”Pokusům se věnoval 9 let. Ze závěrů jeho bádání vznikly tři zákony, dnes známé jako Mendelovy zákony. V roce 1865 odpřednášel výsledky svých pokusů, ohlas na jeho práci byl však minimální. Uznání se dočkal až v roce 1900, kdy byly jeho objevy potvrzeny a Gregor Mendel byl nazván „otcem genetiky“.”*¹

2.2 Základní pojmy genetiky

Alela je konkrétní forma genu. Každý gen je tvořen dvěma alelami, z nichž jedna pochází od otce a druhá od matky. Pro lepší pochopení si uveďme příklad: pokud máme gen určující barvu očí, může jedna alela způsobovat modrou barvu, zatímco druhá může způsobovat barvu zelenou.

Gen je základní jednotka genetické informace.

Genom je kompletní soubor genetické informace organismu, uložený v DNA (nebo RNA u některých virů). Zahrnuje všechny geny i nekódující sekvence.

Genotyp je soubor všech genetických informací organismu, tedy konkrétní kombinace alel, které jedinec zdědil od rodičů.

¹<https://mendelmuseum.muni.cz/o-muzeu/gregor-johann-mendel>

Fenotyp je soubor pozorovatelných vlastností organismu, které jsou výsledkem interakce genotypu s prostředím.

Homozygot je jedinec, který má na sledovaném genu obě alely totožné. Rozlišujeme dominantního homozygota a recesivního homozygota. U dominantního homozygota se gen pro sledovaný znak vyskytuje pouze v dominantních alelách, zatímco u recesivního homozygota pouze v recesivních alelách.²

Heterozygot je jedinec, který má dvě různé alely konkrétního genu.

2.3 1. Mendelův zákon

Zákon o uniformitě F1 (první filiální generace) říká, že při vzájemném křížení dvou homozygotů vznikají potomci, kteří jsou genotypově i fenotypově jednotní. Pokud jde o křížení dvou různých homozygotů, jsou potomci vždy heterozygotními hybridy.

Při křížení dvou homozygotů – dominantního (AA) a recesivního (aa) – vzniká jednotná generace potomků, kteří jsou heterozygotní (Aa) a mají stejný genotyp i fenotyp.

2.4 2. Mendelův zákon

2. Mendelův zákon, neboli zákon o náhodné segregaci genů do gamet, říká, že při křížení dvou heterozygotů může být potomkovi předána každá ze dvou alel (dominantní i recesivní) se stejnou pravděpodobností. Dochází tak k genotypovému a následně i fenotypovému štěpení, tedy segregaci. Pravděpodobnosti pro potomka jsou následující: 25 % pro homozygotně dominantního jedince, 50 % pro heterozygota a 25 % pro homozygotně recesivního jedince. Genotypový štěpný poměr je tedy 1:2:1, zatímco fenotypový štěpný poměr je 3:1, pokud mezi alelami panuje vztah úplné dominance. Při kodominanci odpovídá fenotypový štěpný poměr genotypovému, tedy 1:2:1.³

2.5 3. Mendelův zákon – Zákon o nezávislé kombinovatelnosti alel

Při zkoumání 2 alel současně dochází k téže pravidelné segregaci. Máme-li 2 polyhybridy AaBb může každý tvořit 4 různé gamety (AB, Ab, aB, ab). Při vzájemném křížení tedy z těchto 2 gamet vzniká 16 různých zygotických kombinací. Některé kombinace se ovšem opakují, takže nakonec vzniká pouze 9 různých genotypů (poměr 1:2:1:2:4:2:1:2:1). Nabízí se nám pouze 4 možné fenotypové projevy (dominantní v obou znacích, v 1. dominantní a v 2. recesivní, v 1. recesivní a v 2. dominantní, v obou recesivní). Fenotypový štěpný poměr je 9:3:3:1. Tento zákon samozřejmě platí pouze v případě, že sledované geny se

²<https://www.nzip.cz/rejstrikovy-pojem/4979>

³https://is.muni.cz/el/1441/jaro2010/Bi1BK_ZNT1/um/Mendelovy_zakony_o_dedicnosti.pdf

nachází na různých chromozomech. ⁴

2.5.1 Úplná dominance

Při **úplné dominanci** stačí přítomnost jedné dominantní alely k projevu dominantního znaku. Recesivní alela se v tomto případě fenotypově neprojeví, protože je zcela překryta dominantní alelou. Ve výsledném fenotypovém štěpném poměru se tedy dominantní znak objevuje častěji. V případě dihybridního křížení, kde sledujeme dva znaky s úplnou dominancí, je fenotypový štěpný poměr **9:3:3:1**, protože dominantní znaky se kombinují s recesivními podle pravidel nezávislé kombinovatelnosti alel.

2.5.2 Neúplná dominance

Při **neúplné dominanci** není jedna alela plně dominantní nad druhou, což vede k tomu, že heterozygoti mají **mezičlánekový fenotyp** – tedy znak, který je **kombinací** obou rodičovských fenotypů. Klasickým příkladem je křížení červených a bílých květů hrachoru, kdy heterozygoti mají růžovou barvu. Pokud u dihybridního křížení platí neúplná dominance pro oba sledované znaky, fenotypový štěpný poměr se mění a blíží se poměru **1:2:1:2:4:2:1:2:1**, protože heterozygoti mají odlišný fenotyp od homozygotních jedinců.

2.5.3 Kodominance

V případě **kodominance** se **obě alely plně projeví zároveň** – nevzniká tedy žádný mezičlánekový fenotyp jako u neúplné dominance. Typickým příkladem je **AB krevní skupina**, kde se zároveň projevují jak alela pro antigen A, tak alela pro antigen B. Při dihybridním křížení se kodominantní alely kombinují tak, že fenotypový štěpný poměr se může měnit v závislosti na tom, zda jsou oba sledované geny kodominantní nebo pouze jeden z nich. V takovém případě by se poměr fenotypů mohl lišit od klasického **9:3:3:1** a spíše připomínat **1:2:1**, pokud bychom sledovali jeden znak s kodominancí.

⁴https://www.wikiskripta.eu/w/Základní_zákon_ygenetiky

3 Funkce aplikace Biocross

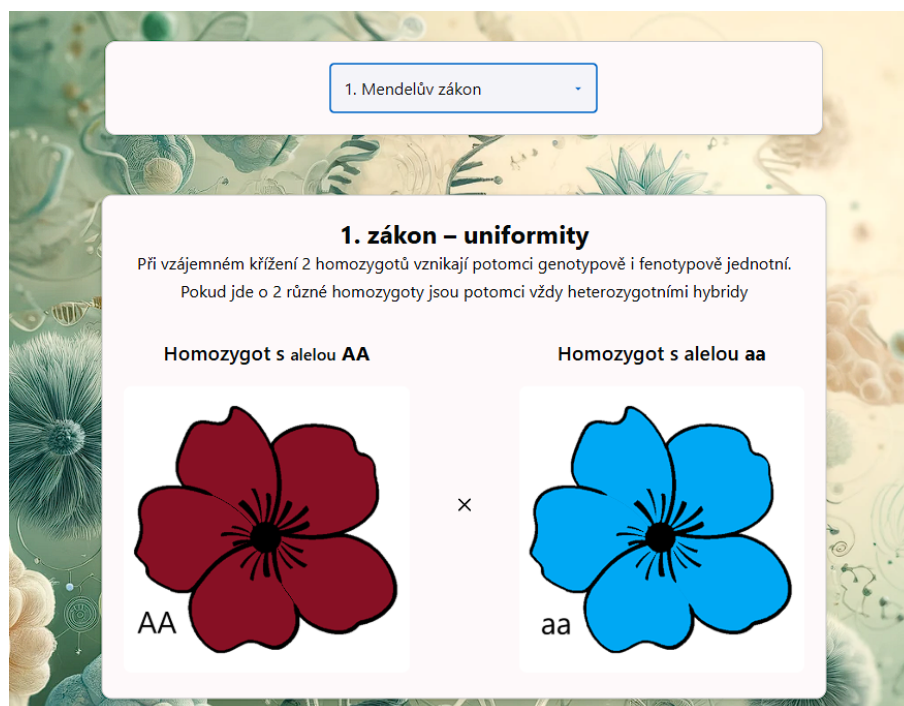
Níže je popis jednotlivých částí mé aplikace, který slouží k lepšímu porozumění její funkčnosti a zároveň poskytuje jasnější představu o celé aplikaci. Popis každé části aplikace je popsán v samostatných kapitolách.

3.1 Domovská stránka

Domovská stránka slouží jako úvodní stránka Biocrossu, jak napovídá její název. Pokud student nezná Mendelovy zákony, nebo si je chce zopakovat, najde zde možnost se je naučit či si látku osvěžit.

3.1.1 Zvolení designu

Při vysvětlování Mendelových zákonů jsem zvolil přístup, který studentům poskytne studijní materiál v moderním a přehledném formátu. Rozhodl jsem se k použití moderního designu a to v podobě kartiček ("flashcards"). Tento způsob prezentace je uživatelsky přívětivý a přehledný.



Obrázek 1: Ukázka kartičky

3.1.2 Složení flashcards

V menu jsou postupně k výběru všechny Mendelovy zákony. Třetí Mendelův zákon je však rozvržen do tří podkategorií, protože jeho vysvětlení zahrnuje různé typy domi-

nance při křížení: **úplná dominance, neúplná dominance a kodominance.**

Tyto typy křížení jsou v dokumentaci rozděleny do jednotlivých podkapitol, protože, jak již bylo zmíněno, vedou k odlišným výsledkům a nelze je jednoduše shrnout do jedné části. Každá podkapitola popisuje specifické rozdíly mezi typy dominance a následné genotypové a fenotypové výsledky.

3.2 Pojmy

Myšlenka stránky *Pojmy* spočívá v tom, že student může rychle získat dodatečné informace, které mu pomohou lépe porozumět výuce.

Tato stránka úzce souvisí s domovskou stránkou, kde probíhá výuka Mendelových zákonů. Z domovské stránky vedou odkazy právě sem, na stránku *Pojmy*. Tyto odkazy zpravidla odkazují na odborné termíny, které by student nemusel znát. Každý termín má zde svou definici a je zde náležitě popsán a vysvětlen.



Obrázek 2: Ukázka stránky Pojmy

Na stránce *Pojmy* jsou vysvětleny pojmy jako je například **alela**, **homozygot** **heterozygot** a další důležité termíny.

Tato stránka je samozřejmě přístupná i mimo domovskou stránku, student se na ni může podívat kdykoliv, kdy by potřeboval. Stránku *Pojmy* může využít i pro přípravu na test, neboť obsahuje přehledně popsané klíčové pojmy a informace, které mu usnadní pochopení a zapamatování si učiva.

3.3 Crossing

Z vlastní zkušenosti jsem měl problém pochopit, jak funguje křížení podle Mendelových zákonů. Proto hlavním cílem stránky Crossing je umožnit studentům lépe si představit, jak samotné křížení probíhá a jaké jsou jeho výsledky.

3.3.1 Co je to Crossing

Crossing je sestaven z hlavního menu, ve kterém student zvolí o jaký typ křížení se jedná. Volit může mezi úplnou dominancí, neúplnou dominancí a kodominancí. Základem je sledování dvou znaků, konkrétně barvy a tvaru, které si student zvolí zcela libovolně. Při změně barvy či tvaru student následně vidí nový výsledek a může pozorovat, jak se výsledky mění. Tento **interaktivní** přístup umožňuje studentům experimentovat s různými variantami, což jim pomáhá lépe pochopit principy jednotlivých typů křížení. Každé křížení funguje jinak a má rozdílné výsledky.

3.3.2 Jak na Crossing

Níže je uvedený postup, jak by měl student postupovat při používání stránky Crossing. Celé rozhraní je navrženo tak, aby bylo intuitivní a snadno pochopitelné.

3.3.2.1 Výběr typu křížení

Prvně student zvolí typ křížení. Vybírá mezi úplnou dominancí, neúplnou dominancí a kodominancí. Jednotlivé typy křížení jsou popsány v kapitole 2 Mendelovy zákony (viz. strana 2).

Každé křížení začíná okýnkem, ve kterém je nadpis, který říká, o který typ se jedná. Vždy pozorujeme dihybridní křížení dvou heterozygotů ($AaBb \times AaBb$), kde "A" představuje dominantní alelu prvního genu (barva), "a" recesivní alelu prvního genu (barva), "B" dominantní alelu druhého genu (tvar), "b" recesivní alelu druhého genu (tvar).

3.3.2.2 Vytvoření vlastních heterozygotů

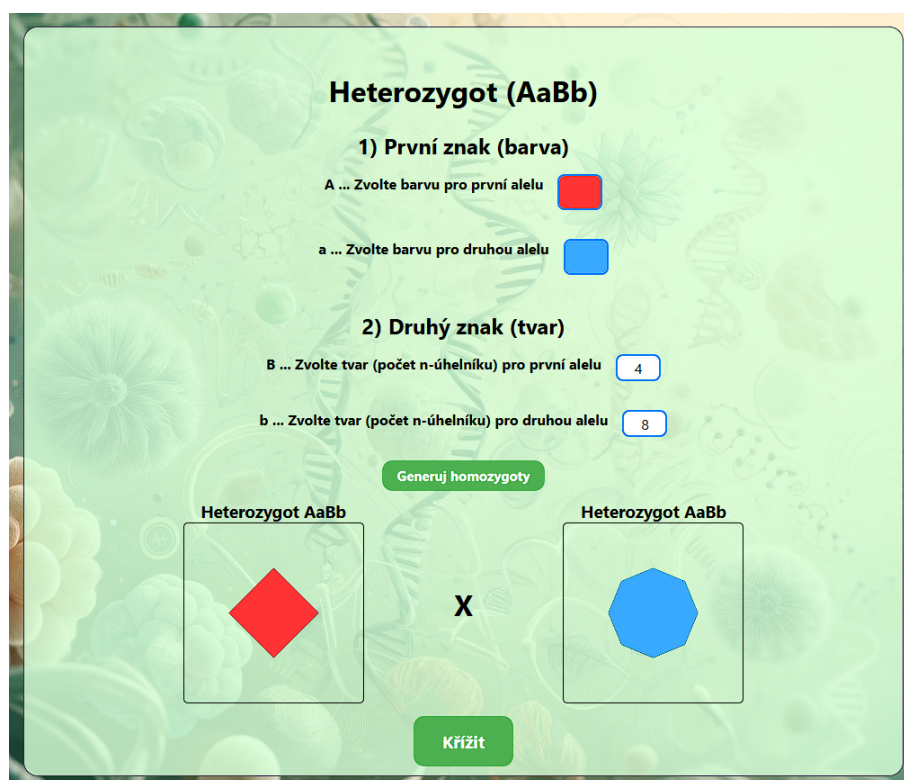
Proces vytváření homozygotů je podobný jak u neúplné dominance, tak u úplné dominance, ale přesto se v jednom ohledu liší. Rozdíl spočívá v tom, že při úplné dominanci si student vybírá, jak budou znaky (barva a tvar) vypadat pro dominantní alelu a pro recesivní alelu. Naopak u neúplné dominance neexistuje jasná dominance ani recesivita – výsledný znak je kombinací obou alel bez vztahu nadřazenosti.

Vytváření heterozygotů popisují pouze pro úplnou dominanci, protože u neúplné dominance se liší jen absencí vztahu nadřazenosti alel, což při samotném vytváření heterozygota nepředstavuje významný rozdíl (zásadní se stává až ve výsledku křížení). Opakovat

tento proces by proto bylo zbytečné.

Kodominance se od těchto dvou typů křížení liší výrazněji, a proto je podrobněji popsána samostatně později v této kapitole.

Student nejprve volí první znak, kterým je barva. Prvně zvolí barvu pro dominantní alelu a následně pro recesivní alelu. U druhého znaku postupuje stejně, avšak místo barvy vybírá tvar, který je definován počtem vrcholů n u n -úhelníku.



Obrázek 3: Příprava křížení

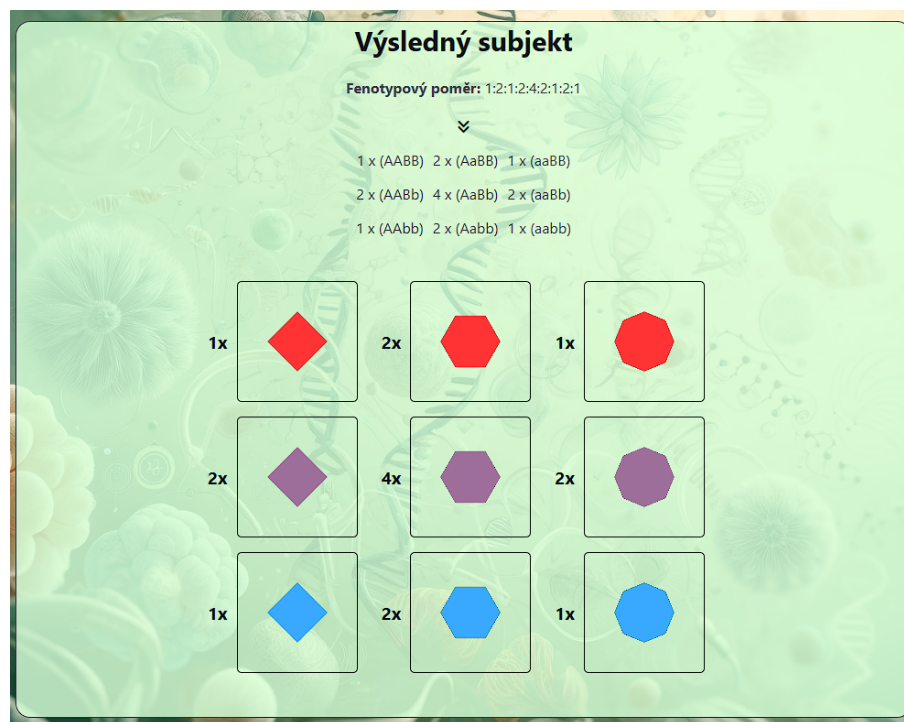
3.3.2.3 Vygenerování připravených homozygotů

Jakmile má student připravené oba znaky, kliknutím na tlačítko "Generuj homozygoty" vygeneruje dva homozygoty odpovídající zvoleným barvám a tvarům. Po vygenerování klikne na tlačítko "Křížit" a proběhne křížení.

3.3.2.4 Výsledek křížení

V okénku s výsledkem křížení je uveden fenotypový poměr vzniklých jedinců. Dále jsou zde rozepsány konkrétní kombinace výsledných alel spolu s jejich četností. Následně jsou zde v rámečcích vizuálně zobrazeny jejich podoby, aby si student dokázal lépe představit výsledky křížení. Díky této vizualizaci se látka snáze naučí a lépe pochopí principy dědičnosti. Ke každému jedinci je připočten odpovídající počet výskytů podle fenotypového poměru, což umožňuje studentovi jasně vidět, kolikrát se daná kombinace

vyskytne.



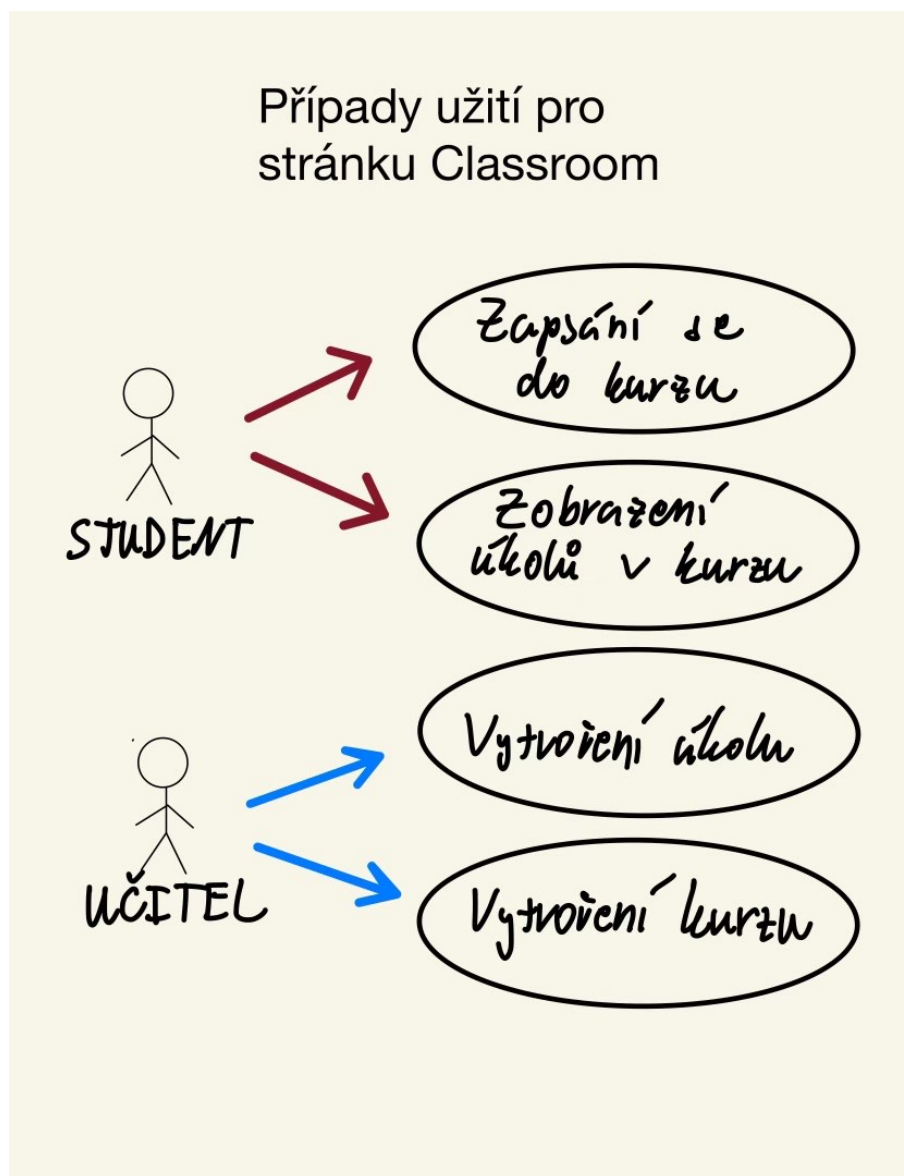
Obrázek 4: Výsledek křížení

3.3.3 Více variant jak měl Crossing vypadat

Stránka Crossing prošla několika designovými úpravami, protože navrhnout vzhled tak, aby byl co nejvíce uživatelsky přívětivý, bylo náročným úkolem.

3.4 Classroom

Funkce Classroom se dá rozdělit do dvou případů užití. První případ je použití jako student, druhý jako učitel. Princip této stránky je podobný aplikaci Google Classroom od společnosti Google.



Obrázek 5: Use Case pro stránku Classroom

3.4.1 Použití jako student

3.4.1.1 Přihlášení se do kurzu

Student má na začátku možnost přihlásit se do kurzu prostřednictvím formuláře, kde zadá kód kurzu. Tento kód získá od svého učitele. Po zadání kódu je student automaticky přihlášen do příslušného kurzu, což mu umožňuje přístup k zobrazení úkolů a následné a odevzdání domácích úkolů, které může zpracovat na stránce *Crossing* a výsledek si uložit

právě do PDF formátu.

3.4.1.2 Zobrazení všech úkolů

V sekci *Úkoly* má student přehledně zobrazen seznam všech úkolů, které má splnit. Tento seznam obsahuje základní informace o úkolu a to název úkolu, do jakého kurzu úkol spadá a který učitel úkol zadal. Na každý úkol se dá kliknout a po kliknutí je student přesměrován do konkrétního kurzu, kam úkol spadá.

3.4.1.3 Odevzdání úkolu v kurzu

Každý úkol má možnost odevzdání, kde student nahraje stáhnutý soubor PDF formátu ze stránky *Crossing*. Po odevzdání úkolu je studentovi zobrazeno potvrzení o úspěšném odevzdání.

3.4.2 Použití jako učitel

3.4.2.1 Vytvoření kurzu

Učitel má podobně rozvržené prostředí jako student. V sekci *Vytvořit kurz* může učitel jednoduše založit nový kurz, pro který se vygeneruje unikátní kód. Tento kód předá studentům, kteří se podle něj do kurzu přihlásí.

3.4.2.2 Vytvoření úkolu

V sekci *Vytvořit úkol* může učitel vytvářet nové úkoly. Každý úkol obsahuje termín odevzdání, termín zadání, název úkolu, popis úkolu a do jakého kurzu úkol patří.

3.4.2.3 Zobrazení odevzdaných úkolů v kurzu

V sekci *Kurzy* má učitel přehled o všech kurzech, které vyučuje. Po rozkliknutí konkrétního kurzu se učiteli zobrazí přehled zadaných úkolů. V každém úkolu si může učitel zobrazit všechny odevzdané úkoly, které studenti vložili. Odevzdaný úkol se skládá ze studentova jména a příslušné přílohy (PDF souboru).

4 Design aplikace

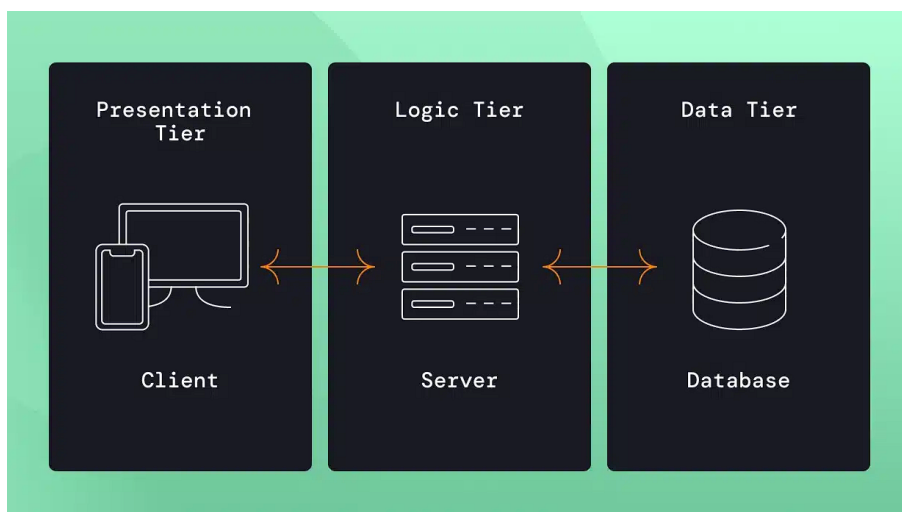
4.1 Architektura

Aplikace Biocross je postavena na principu **3-layer architecture** (třívrstvé architektury). Tato architektura rozděluje aplikační logiku, zobrazení a databázi do samostatných vrstev, což usnadňuje správu a úpravy systému v průběhu času, protože každá vrstva se dá upravovat nezávisle na ostatních. Jakmile aplikaci vytvoříme pomocí této architektury, při dalším vývoji – například pro mobilní zařízení nebo jinou platformu – stačí napsat pouze novou prezentační vrstvu, ale aplikační logika a datová vrstva mohou zůstat beze změny.

Presentation Layer – prezentační vrstva, která slouží k zobrazení dat uživateli. Šablony jsou napsané v **EJS** (Embedded JavaScript), což je šablonovací jazyk, který rozšiřuje HTML o dynamické prvky, jako jsou proměnné a podmíněné výrazy.

Logic Layer – vrstva aplikační logiky, která obsahuje veškeré zpracování dat a logiku. Zahrnuje například výpočty, databázové dotazy a další operace spojené s daty. Aplikační logiku je psána v **Node.js**. Endpointy aplikace komunikují s MySQL databází pomocí knihovny *mysql2*.

Data Layer – datová vrstva je zodpovědná za správu a uchovávání dat. V mém případě používám **MySQL**, což je relační databázový systém, který umožňuje správu dat pomocí strukturovaných dotazů. MySQL jsem zvolil, jelikož s ním mám také zkušenost již z druhého ročníku. Také se mi s ním dobře pracovalo a je to jedna z možných variant pro práci v Node.js.



Obrázek 6: Třívrstvá architektura

4.2 Prezentační vrstva (Frontend)

Jak jsem již zmínil, uživatelské rozhraní (frontend) jsem vytvořil v **EJS** (Embedded JavaScript), což umožňuje dynamické generování HTML. Vzhled stránek jsem navrhl pomocí **CSS** (kaskádových stylů) a interaktivní prvky jsem implementoval v **JavaScriptu**. Pro JavaScript existují alternativy, jako například Dart nebo Elm, avšak zvolil jsem JavaScript kvůli jeho širokému rozšíření a mé předchozí zkušenosti s ním.

EJS umožňuje vkládat do HTML šablon například proměnné, do kterých se z backendu posílají konkrétní hodnoty, jako například jméno přihlášeného uživatele. Díky této funkci lze snadno generovat dynamický obsah a přizpůsobovat stránky podle aktuálních dat. Funkce EJS jsem ve své aplikaci využil jen minimálně.

Nejprve vysvětlím základní principy, jaké jsem použil při vytváření frontend vrstvy a poté implementaci tří klíčových funkcionalit na stránce *Crossing*.

4.2.1 Principy

4.2.1.1 Design kartiček

Většina designu aplikace je založena na rozvržení do samostatných bloků, které připomínají bubliny nebo kartičky (flashcards). Tento design je jednotný pro většinu stránek, jako je **Crossing**, **Pojmy** nebo **Domovská stránka**. Proto logiku popíšu na jedné vybrané stránce, konkrétně na domovské stránce, kde se student učí Mendelovy zákony prostřednictvím interaktivních flashcards.

Struktura těchto bublin je navržena tak, že jsou umístěny uvnitř kontejneru s třídou `body-flashcard`. Tento kontejner obsahuje všechny kartičky s výukovým obsahem a zajišťuje jejich správné zobrazení. Níže je ukázka kódu pro definici tohoto kontejneru:

```
.body-flashcard {  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  padding: 20px;  
}
```

Jednotlivé kartičky (flashcards) jsou vytvořeny pomocí kontejneru s třídou `flashcard`. Tento kontejner definuje vzhled a rozložení obsahu kartiček tak, aby bylo zajištěno správné zobrazení. Stylování zahrnuje vlastnosti jako barvu pozadí, ohraničení, zaoblené rohy a stín, který přidává jemný vizuální efekt. Dále je nastavena animace pro interaktivní efekt při najetí myši.

```
.flashcard {
```

```

background-color: #FEF9FA;
border: 1px solid #ccc;
border-radius: 10px;
padding: 20px;
margin: 10px 0;
width: 100%;
max-width: 50rem;
box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
transition: transform 0.2s;
text-align: center;
color: #060D0D;
}

.flashcard:hover {
    transform: scale(1.01);
}

```

Efekt `transform: scale(1.01);` zajistí, že při najetí myši na kartičku se její velikost nepatrně zvětší (o 1 %), což vytváří příjemnou vizuální interakci.

V některých případech jsou objekty v kartičkách uspořádány do vertikálních nebo horizontálních kontejnerů, například po dvojicích nebo trojicích. Příklad: máme obrázek s výsledným heterozygotem, jeho jméno a znázornění jeho alel pro dané znaky. Aby byly tyto prvky správně zarovnány pod sebe a zobrazily se přehledně, jsou umístěny do kontejneru `vertical-div`. Dva takové kontejnery mohou být umístěny vedle sebe uvnitř `horizontal-div`, který zajišťuje jejich správné rozložení vedle sebe, horizontálně. Tímto způsobem je zajištěno, že prvky budou správně zarovnané.

```

.horizontal-div {
    display: flex;
    margin-top: 2rem;
    justify-content: center;
    gap: 40px;
}

.vertical-div {
    display: flex;
    flex-direction: column;
    align-items: center;
    margin-top: 2rem;
    gap: 20px;
}

```


4.2.1.2 Responzivní design

Aby byl design aplikace responzivní, používám pravidlo **media query** ve formátu `@media` (podmínka). Toto pravidlo zajišťuje přizpůsobení vzhledu aplikace různým zařízením, zejména při změně velikosti obrazovky. Jako příklad jsem použil kontejner `flashcard`, kde **media query** upravuje šířku každé kartičky v závislosti na šířce okna prohlížeče. Níže je ukázka kódu:

```
/* Pro tablety a vetsi obrazovky */
@media (min-width: 768px) {
    .flashcard {
        width: 80%; /* Uprav sirku pro tablety */
    }
}

/* Pro stolni pocitace a vetsi obrazovky */
@media (min-width: 1024px) {
    .flashcard {
        width: 60%; /* upraví sirku pro pocitace */
    }
}
```

4.2.1.3 Template inclusion

Šablony rozdělují na menší opakovaně použitelné části, což mi umožňuje vyhnout se zbytečnému opakování stejného HTML kódu. Tento přístup, nazývaný **template inclusion**, umožňuje znovupoužití předem definovaných částí šablon, což usnadňuje údržbu a zlepšuje organizaci kódu. Má velkou výhodu zejména v situacích, kdy potřebujeme změnit opakující se část kódu, která se nachází na více stránkách aplikace. Stačí provést úpravu pouze v jednom souboru, který tuto část obsahuje, a změna se automaticky projeví na všech místech, kde je tento soubor použit. To nám šetří čas a eliminuje potřebu upravovat stejný kód ve více souborech. Typickým příkladem je navigační panel (navbar), který vytvoříme jako samostatnou šablonu a následně jej vložíme do všech relevantních stránek aplikace. Tím zajistíme jednotný vzhled a snadnou správu kódu.

Nejčastěji tento přístup využívám pro **navigační panel** (navbar) a **patičku stránky** (tail).

```
<body>
  <%- include('navbar.ejs') %>
  <div class="select-krizeni">...</div>
  <!-- UPLNA DOMINANCE -->
  <div class="body-crossing" id="uplnaDominance">...</div>
  ...
  <%- include('../views/footer.ejs') %>
</body>
```

4.2.2 Crossing a křížení

Křížení na stránce Crossing je udělané pomocí JavaScriptu. Používám několik důležitých funkcí, které následně vysvětlím a popíšu: vytváření a malování polygonů, výpočet počtu vrcholů u vytvořených n -úhelníků a mixování barev.

4.2.2.1 Práce s barvami

Pro práci s barvami používám tři funkce. Používám funkci `hexToRgb(hex)` k převodu hexadecimálního zápisu barvy na RGB složky. Odstraní znak # a převede jednotlivé části na číselné hodnoty.

Funkci `rgbToHex(r, g, b)` používám k opačnému převodu, tedy z RGB složek zpět na hexadecimální formát. Každá složka se převede na šestnáctkový formát a zároveň na dvě číslice.

Pro míchání dvou barev používám funkci `mixColors(color1, color2)`, která převede hexadecimální kódy na RGB, vypočítá průměr složek a výsledek vrátí zpět jako hexadecimální kód.

```
function mixColors(color1, color2) {  
  const rgb1 = hexToRgb(color1);  
  const rgb2 = hexToRgb(color2);  
  
  const r = Math.round((rgb1.r + rgb2.r) / 2);  
  const g = Math.round((rgb1.g + rgb2.g) / 2);  
  const b = Math.round((rgb1.b + rgb2.b) / 2);  
  
  return rgbToHex(r, g, b);  
}
```

Ukázka kódu 1: Funkce pro míchání barev-mixColors(color1, color2)

4.2.2.2 Vytváření heterozygotů

U křížení jsou tři typy dominance a každá z nich má svou vlastní funkci na vytvoření výsledků.

Pro malování n -úhelníků jsem použil HTML prvek `<canvas>`. `<canvas>` je HTML prvek, který se používá k vykreslování grafiky pomocí JavaScriptu. Poskytuje oblast, na kterou lze kreslit různé tvary, obrázky nebo grafy. Tento prvek je podporován většinou moderních prohlížečů, což je velká výhoda.

Nejprve se načte počet vrcholů pro první a druhý n-úhelník (heterozygoty).

```
var N_Uhelnik1 = parseInt(document.getElementById('input1').value);
```

Ukázka kódu 2: Načtení hodnoty n-úhelníku

Poté se načte canvas, do kterého se dané n-úhelníky vykreslí. Je nutné vymazat jeho obsah, aby v nich nezůstaly "zbytky" z předchozího křížení. Metoda `getContext` zpřístupňuje metody pro vykreslování. Já použil konkrétně typ "2d", který se používá pro vykreslování 2D grafiky.

```
const canvas1 = document.getElementById('canvas1');  
const ctx1 = canvas1.getContext('2d');  
ctx1.clearRect(0, 0, canvas1.width, canvas1.height);
```

Ukázka kódu 3: Inicializace canvasu a vymazání obsahu

Po načtení objektu "canvas" se načte barva z palety barev. Jak vytvořit funkční paletu barev jsem zjistil na webu <https://medium.com/@laners.org/how-to-add-a-color-picker-in-html-ebd55e419b16>⁶

```
const colorPicker1 = document.getElementById('colorPicker1');  
const selectedColor1 = colorPicker1.value;
```

Ukázka kódu 4: Načtení barvy pro n-úhelník

Tento kód následně nakreslí n-úhelník a vyplní ho zvolenou barvou

```
drawPolygon(ctx1, 200, 200, 100, N_Uhelnik1);  
ctx1.fillStyle = selectedColor1;  
ctx1.fill();
```

Ukázka kódu 5: Vykreslení n-úhelníku

4.2.2.3 Samotné křížení

Ukázku kódu křížení by bylo nejlepší ukázat na neuplné dominanci a to kvůli její variabilitě výsledků. Ve své podstatě jde o generování zmíněných n-úhelníků, ale některé z nich mají smíchané barvy nebo počet vrcholů a to právě podle 3. Mendelova zákona.

Postup je stejný jako v podkapitole vytváření heterozygotů. Jelikož při neúplné dominanci dochází k projevení obou znaků, bylo nutné upravit barvu a počet vrcholů u některých n-úhelníků. Barvu upravuji tak, že první barva se smíchá s druhou. Smíchání barev zajišťuje funkce `mixColors(color1, color2)`. Počet vrcholů u n-úhelníků upravuji tak, že se vypočítá aritmetický průměr z počtu vrcholů prvního a druhého heterozygota.

⁶<https://medium.com/@laners.org/how-to-add-a-color-picker-in-html-ebd55e419b16>

Funkce pro smíchání barev

```
function mixColors(color1, color2) {  
  const rgb1 = hexToRgb(color1);  
  const rgb2 = hexToRgb(color2);  
  
  const r = Math.round((rgb1.r + rgb2.r) / 2);  
  const g = Math.round((rgb1.g + rgb2.g) / 2);  
  const b = Math.round((rgb1.b + rgb2.b) / 2);  
  
  return rgbToHex(r, g, b);  
}
```

Výpočet aritmetického průměru počtu vrcholů

```
var smisenyN_Uhelnik = (N_Uhelnik1 + N_Uhelnik2) / 2;
```

Po připravení veškerých potřebných parametrů na vytvoření všech různých heterozygotů se n-úhelníky vykreslí. Tento postup je již popsán v předchozí podkapitole.

4.3 Vrstva aplikační logiky (Backend)

Pro vrstvu aplikační logiky jsem použil Node.js, což je free, open-source, cross-platform JavaScript runtime environment.⁷ Node.js se primárně používá pro tvorbu serverové části webových aplikací. Node.js lze také použít pro tvorbu aplikací na PC nebo mobilní telefony tablety... Díky Node.js se dá v JavaScriptu pracovat nejen v prostředí prohlížeče, ale také na serveru.

Moduly v Node.js: Node.js pracuje s **moduly**, které fungují podobně jako knihovny v jiných programovacích jazycích. Existují **vestavěné moduly** (*core modules*), které jsou součástí Node.js a není nutné je instalovat. Jedním z nich je modul *http*, který je nezbytný pro vytvoření HTTP serveru.

Balíčkový systém npm: **npm** (Node Package Manager) je správce balíčků pro Node.js. Umožňuje instalovat, spravovat a aktualizovat knihovny a závislosti v projektu.

Při vytváření aplikace **Biocross** jsem použil framework **Express.js**, který se používá při vývoji webových aplikací v **Node.js**. Díky Expressu je tvorba webové aplikace rychlejší a jednodušší, protože například není nutné ručně zpracovávat HTTP metody – Express umožňuje implementovat stejnou funkcionalitu výrazně jednodušeji. Poskytuje také snadný způsob přidávání funkcí, jako je autentizace, logování nebo zpracování JSON dat, prostřednictvím **middleware** funkcí. Dále jsem jej zvolil, protože podporuje šablonovací jazyky, jako je právě **EJS**.

4.3.1 Vytvoření kostry a základního nastavení aplikace

Framework Express obsahuje generátor, který dokáže vygenerovat kostru aplikace. Pro spuštění generátoru pomocí *npx* (od Node.js 8.2.0+) použijte příkaz:

```
npx express-generator
```

Ukázka kódu 6: Spuštění generátoru aplikace

Následující příkaz vytvoří Express aplikaci s názvem Biocross. Aplikace bude umístěna do složky *Biocross* v aktuálním pracovním adresáři a jako šablonovací engine bude použito EJS:

```
$ express --view=ejs Biocross
```

Ukázka kódu 7: Vytvoření kostry aplikace pomocí generátoru aplikace

4.3.1.1 Start-up skript a knihovna Nodemon

Zajímavou součástí aplikace je **startup script**, který slouží k jejímu spuštění. Tento skript je definován v souboru *package.json*, který obsahuje informace o balíčcích a konfi-

⁷<https://nodejs.org>

guraci projektu.

Důležitou roli zde hraje knihovna **nodemon**⁸, která automaticky restartuje aplikaci při jakékoliv změně v kódu, což vývoj výrazně usnadňuje a zrychluje, protože není nutné ji spouštět ručně po každé úpravě.

```
{
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "ejs": "^3.1.10",
    "express": "~4.16.1"
  }
}
```

Ukázka kódu 8: Konfigurace v souboru package.json

Skript pro spuštění aplikace s automatickým restartem pomocí knihovny *Nodemon* lze spustit tímto příkazem:

```
npm run dev start
```

Ukázka kódu 9: Příkazy pro spuštění aplikace

4.3.1.2 Základní nastavení aplikace

Ze začátku je nutné provést základní nastavení aplikace. Následující kód ukazuje, jak správně nakonfigurovat webovou aplikaci pomocí frameworku Express.

```
var express = require('express');
var app = express();

app.set('views', 'views')
app.set('view engine', 'ejs');

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use(express.static('public'));

const PORT = process.env.PORT || 3000;
```

⁸<https://nodemon.io>

Ukázka kódu 10: Základní konfigurace aplikace v Express.js

Tento kód definuje několik klíčových nastavení a pár z nich níže popisují.

- `require('express')` – Načte modul Express.js.
- `app.set('views', 'views')` – Určuje složku, ve které se nacházejí šablony pro vykreslování HTML stránek.
- `app.set('view engine', 'ejs')` – Definuje šablonovací engine.
- `app.use(express.json())` – Povoluje zpracování JSON dat v požadavcích.
- `app.use(express.static('public'))` – Nastavuje složku *public* jako zdroj pro statické soubory (obrázky, JavaScript, CSS).

4.3.2 Principy

4.3.2.1 Routes v Node.js

V Node.js se pro lepší přehlednost projektu používají „routes“ (směrování). Každá „ruta“ (URL adresa) je propojena s konkrétní funkcí nebo handlerem, který zpracovává příchozí požadavky.

Obvykle má každá významná část aplikace svůj vlastní soubor v adresáři */routes*, kde jsou definovány způsoby zpracování různých HTTP metod a typů požadavků. Rozdělení logiky do samostatných souborů pomáhá udržet projekt strukturovaný a srozumitelný.

V hlavním souboru aplikace **app.js** se jednotlivé „routy“ načítají a registrují, například pro stránku *Classroom*:

```
const classroomRouter = require("./routes/classroom");
app.use("/classroom", classroomRouter);
```

Ukázka kódu 11: Registrace routeru v app.js

Tento kód říká, že všechny požadavky směřující na */classroom* budou zpracovávány v souboru *routes/classroom.js*.

V souboru *routes/classroom.js* je pak definována samotná logika pro tuto část aplikace:

```
1 const express = require('express');
2 const router = express.Router();
3
4 router.post("/", (req, res) => {
5   // Pridani studenta do databaze
```

```

6 });
7
8 module.exports = router;

```

Ukázka kódu 12: Definice routeru v `routes/classroom.js`

Je důležité náš *router* exportovat, aby jej bylo možné použít v `app.js`.

4.3.2.2 HTTP metody

Jak už jsem zmínil, Express.js nám pomáhá s rychlým zpracováním HTTP požadavků, jako jsou například **GET**, **POST**, **PUT** nebo **DELETE**. Níže je ukázka kódu, který řeší HTTP požadavek na URL `/login`.

```

app.get("/login", (req, res) => {
  res.render("login");
});

```

Ukázka kódu 13: Zpracování HTTP GET požadavku na `/login`

Tento endpoint reaguje na HTTP požadavek typu **GET** zaslaný na adresu `/login`. Když uživatel provede požadavek na tuto adresu, aplikace spustí odpovídající funkci, která:

1. Přijme objekt `req` obsahující informace o požadavku.
2. Přijme objekt `res` sloužící k odeslání odpovědi.
3. Použije metodu `res.render("login")`, která vygeneruje HTML stránku na základě šablony `login.ejs` a odešle ji zpět uživateli.

Díky tomu se po odeslání odpovědi zobrazí na adrese `/login` přihlašovací stránka.

V aplikaci používám HTTP požadavek typu **POST** především při práci s databází, nejčastěji pro ukládání dat například při registraci nového uživatele, vytvoření nového kurzu nebo přihlášení se do existujícího kurzu.

Následující kód ukazuje, jak POST požadavek zpracovat.

```

router.post('/', (req, res) => {
  if ('login' === req.body.formType) {
    // Zpracovani prihlaseni uzivatele
    ...
  } else if ('register' === req.body.formType) {
    ...
    // Ulozeni noveho uzivatele do DB
    // Presmerovani na login stranku
  }
});

```



```

        res.redirect('/login');
    }
});

```

Ukázka kódu 14: Zpracování POST požadavku v Express.js

Po vyplnění registračního formuláře a jeho odeslání se odešle požadavek typu POST na adresu */login*. Tento formulář musí být v HTML kódu zabalen do tagu `<form>` s příslušnou akcí.

```

<form class="register-form" action="/login" method="POST">
    ...
</form>

```

Ukázka kódu 15: Registrační formulář

Server následně tento POST požadavek zpracuje, vyhodnotí jeho obsah a podle toho se zachová. Při úspěšné registraci uloží nového uživatele do databáze a přesměruje uživatele na přihlašovací stránku */login*.

4.3.2.3 Zpracování různých formátů odpovědí v Node.js

Pro odeslání konkrétních dat z databáze (například seznam kurzů, do kterých je student přihlášen) využívám funkci *res.json()*. Tato funkce umožňuje odeslat data načtená z databáze ve formátu JSON jako odpověď na požadavek klienta. Funkce *res.json(data)* převede data do formátu JSON a odešle je na frontend. Na straně frontendu lze pomocí funkce *fetch()* získat data ze serveru, zpracovat je a zobrazit například na webové stránce.

Na straně serveru se pro získání dat z databáze využívá funkce *getCurses.getCurses()*, která načte seznam kurzů přihlášeného uživatele. Tato data se poté odešlou klientovi buď ve formátu HTML, nebo JSON, podle toho, jaký formát klient požaduje:

```

const getCurses = require('./routes/methods/getCurses');

app.get("/classroom", async(req, res) => {
    // nacte existujici kurzy uzivatele
    var arrCurses = await getCurses.getCurses();

    try {
        res.format({
            html: async () => {
                res.render("classroom");
            },
            json: () => {
                res.json(arrCurses);
            }
        });
    }
});

```

```

    });
  } catch (err) {
    console.error("Chyba pri nacitani kurzu:", err);
    res.status(500).send("Chyba serveru");
  }
});

```

Ukázka kódu 16: Načítání kurzů na backendu

Na straně klienta lze pomocí *fetch()* získat seznam kurzů ze serveru a zobrazit je na stránce:

```

fetch(`/classroom` + location.search, { headers: {
  'Accept': 'application/json'
}})
  .then(response => response.json())
  .then(courses => displayCurses(courses))
  .catch(error => console.error(error));

function displayCurses(courses) {
  divCurses.innerHTML = '';
  courses.forEach(result => {
    const listOfCurses = document.createElement('div');
    listOfCurses.innerHTML = '<li><a href="#">'
    + result.course_className + " "
    + result.course_teacherName + " "
    + result.course_year+'</a></li>';
    divCurses.appendChild(listOfCurses);
  });
}

```

Ukázka kódu 17: Načítání a zobrazení kurzů na frontendu

4.3.3 Seznam Endpointů

V tabulce je seznam dostupných endpointů s jejich názvem, typem a krátkým popisem.

Název	Typ	Popis
/	GET	Vyrendruje Domovskou stránku
/login	GET	Vyrendruje login stránku
/login	POST	Zaregistruje nebo přihlásí uživatele
/crossing	GET	Vyrendruje stránku Crossing
/pojmy	GET	Vyrendruje stránku Pojmy
/classroom	GET	Vyrendruje Classroom pro studenta
/classroom	POST	Zapíše studenta do kurzu
/classroomUcitel	GET	Vyrendruje Classroom pro učitele
/classroomUcitel/novyKurz	GET	Vyrendruje stránku pro tvorbu nového kurzu
/classroomUcitel/novyKurz	POST	Vytvoří nový kurz
/classroomUcitel/ukoly	GET	Vyrendruje stránku pro tvorbu nového úkolu
/classroomUcitel/ukoly	POST	Vytvoří nový úkol

Tabulka 1: Seznam endpointů

4.4 Datová vrstva (Database)

4.4.1 Použitá databáze

Pro aplikaci Biocross jsem použil strukturovanou relační databázi **MySQL**. MySQL je systém pro správu relačních databází (RDBMS), který využívá strukturovaný dotazovací jazyk SQL (Structured Query Language) pro práci s daty.

4.4.2 Schéma databáze

Databázová struktura aplikace obsahuje několik tabulek, které uchovávají informace o studentech, učitelích, kurzech a úkolech.

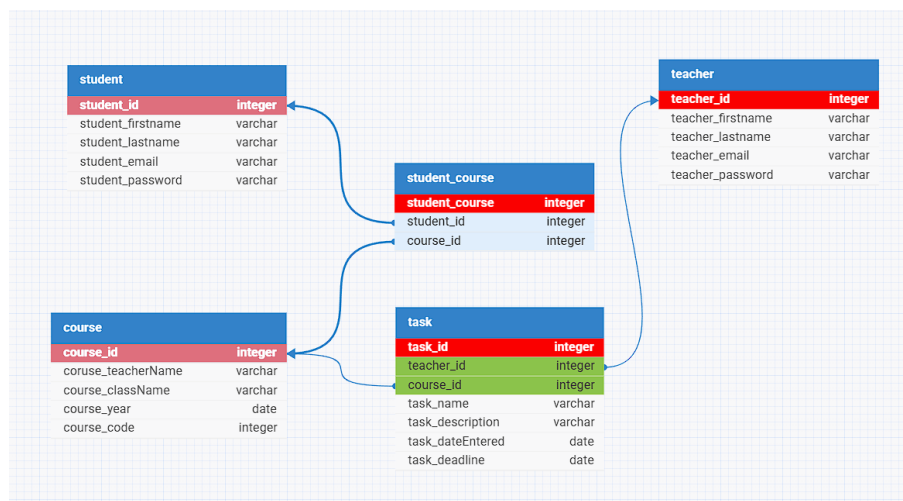
Tabulka **student** uchovává informace o studentech, jako jsou jejich jména, e-maily a hesla.

Tabulka **course** obsahuje informace o kurzech, včetně názvu, učitele a roku konání.

Tabulka **teacher** uchovává informace o učitelích, včetně jejich jmen, e-mailů a hesel.

Tabulka **student_course** propojuje studenty s kurzy prostřednictvím cizích klíčů **student_id** a **course_id**.

Tabulka **task** obsahuje informace o úkolech, které jsou přiřazeny ke kurzům a učitelům. Je propojena s tabulkami **teacher** a **course** pomocí cizích klíčů **teacher_id** a **course_id**.



Obrázek 7: Schéma databáze

4.4.3 Instalace databázové knihovny

Pro komunikaci s MySQL databází v Node.js používám knihovnu `mysql2`. Instalaci provedu pomocí balíčkovacího systému `npm`:

```
npm install mysql2
```

Ukázka kódu 18: Instalace knihovny `mysql2`

4.4.4 Připojení k databázi

Pro připojení k MySQL databázi v Node.js slouží následující kód:

```

1 var mysql = require('mysql2');
2 var con = mysql.createConnection({
3   host: '127.0.0.1',
4   port: '3306',
5   user: 'root',
6   password: '****',
7   database: 'biocross'
8 });
9 con.connect(function(err) {
10   if (err) throw err;
11   console.log('Database is connected successfully !');
12 });
13 module.exports = con;
  
```

Ukázka kódu 19: Připojení k MySQL databázi

4.4.5 Příklad použití

Níže je ukázka řešení, jak načíst z databáze seznam kurzů pro daného uživatele:

```
return new Promise((resolve, reject) => {
  var sql = "SELECT course_teacherName, course_className,
    course_year FROM student_course sc
    JOIN student s ON(sc.student_id=s.student_id)
    JOIN course c ON(sc.course_id=c.course_id)
    WHERE s.student_id = 1;"
  db.query(sql, function (err, result, fields) {
    if (err) {
      console.log(err);
      reject(err);
    }
    resolve(result);
    // console.log(result);
  });
});
```

Ukázka kódu 20: Načtení kurzů z databáze

5 Deployment

5.1 Deployment do hostovacích center

Aplikace Biocross je hostována na stránce Render.com⁹. Aplikace běží na doméně Biocross.cz Databáze je hostovaná na stránce Aiven.io¹⁰

The screenshot shows the 'Source Code' tab of a new service on Render.com. The configuration fields are as follows:

- Source Code:** gyarab / 2024-4e-maxa-Biocross (with an 'Edit' link)
- Name:** 2024-4e-maxa-Biocross (with a note: 'A unique name for your web service.')
- Project:** Optional. Add this web service to a project once it's created. A button '+ Create a project' is visible.
- Language:** Node (dropdown menu)
- Branch:** main (dropdown menu)
- Region:** Frankfurt (EU Central) (dropdown menu, with a note: '1 existing service'). A link 'Deploy in a new region' is also present.
- Root Directory:** Optional. If set, Render runs commands from this directory instead of the repository root. (Text input field with 'e.g.: src')
- Build Command:** \$ npm ci (text input field)
- Start Command:** \$ node app.js (text input field)

Obrázek 8: Deployment na stránce Render.com

5.2 Instalace v lokálním prostředí

Pokud si chcete aplikaci stáhnout a spustit na svém lokálním počítači, postupujte podle následujícího návodu. Tento návod vás provede procesem instalace závislostí a spuštěním aplikace.

5.2.1 Požadavky

Před instalací se ujistěte, že máte nainstalované následující komponenty:

- Node.js (doporučená verze: 20+)

⁹<https://render.com/>

¹⁰<https://aiven.io/>

- NPM (součástí instalace Node.js)
- Webový prohlížeč podporující HTML5 (např. Google Chrome, Mozilla Firefox)

5.2.2 Stažení aplikace

Zdrojový kód aplikace si můžete naklonovat pomocí příkazu:

```
git clone https://github.com/gyarab/2024-4e-maxa-Biocross.git
```

Ukázka kódu 21: Stažení aplikace

Poté přejděte do stažené složky:

```
cd 2024-4e-maxa-Biocross
```

Ukázka kódu 22: Přejít do složky projektu

5.2.3 Instalace závislostí

Jakmile budete v této složce, pro správné fungování aplikace je potřeba nainstalovat všechny závislosti. Tento příkaz instaluje balíček a všechny jeho závislosti.

```
npm install
```

Ukázka kódu 23: Instalace závislostí

5.2.4 Spuštění aplikace

Aplikaci spustíte jedním z následujících příkazů:

```
npm start  
npm run dev start //spusteni s knihovnou nodemon
```

Ukázka kódu 24: Spuštění aplikace

Po spuštění by měla být aplikace dostupná na lokální adrese **http://localhost:3000**. Otevřete tuto adresu ve svém webovém prohlížeči a můžete aplikaci začít používat.

6 Závěr

Cílem mé práce bylo vytvořit webovou aplikaci, která umožní lépe pochopit výuku Mendelových zákonů prostřednictvím jejich vizualizace a následného procvičení. Vizualizace pomocí obrázků a výukových informací prezentovaných v přehledném designu *flashcards*, následné procvičení na stránce *Crossing* společně se zadáním úkolů na stránce *Classroom* umožňují studentům lépe pochopit, jak křížení funguje. Díky této kombinaci vizualizace a aktivního procvičování se studenti mohou Mendelovy zákony naučit snáze a proto si myslím, že má práce splnila zadání.

Tato práce mě naučila přistupovat k problémům nejen z hlediska technického řešení, ale i z pohledu designu, který musí být nejen vizuálně přitažlivý, ale také funkční a intuitivní pro uživatele. Dalším významným přínosem pro mě bylo vytvoření dobře strukturované dokumentace.

6.1 Budoucí rozvoj

V rámci dalšího vývoje tohoto projektu plánuji zdokonalení *Classroomu*, zejména možnost exportu výsledků crossingu do PDF formátu, aby je studenti mohli snadno odevzdat učiteli. Dalším cílem je také optimalizace zobrazení aplikace na mobilních zařízeních, aby byla plně responzivní a dobře použitelná i na menších obrazovkách. Dále bych rád, aby byla moje aplikace otestována přímo studenty, kteří by mi mohli poskytnout zpětnou vazbu o tom, jak jim pomohla při učení. Tento feedback by mohl přispět k dalšímu vylepšení aplikace a jejímu lepšímu přizpůsobení potřebám uživatelů.

7 Speciální citace

Při tvorbě této práce jsem čerpal informace z několika zdrojů:

- Pro práci v **Node.js** a **Express.js** jsem čerpal ze své ročníkové práce z třetího ročníku, která je dostupná na GitHubu: <https://github.com/KrystofMaxaCZ/2023-3e-Aktivitar.git>.
Také jsem využil knihu: **Shelly. 2016. Learning Node. 2. vydání. Sebastopol: O'Reilly Media. ISBN 978-1-4919-4311-3.**
- K tvorbě vysvětlení Mendelových zákonů jsem využil následující online zdroje:
 - WikiSkripta – Základní zákony genetiky:
https://www.wikiskripta.eu/w/Zakladni_zakony_genetiky
 - Materiály Masarykovy univerzity – Mendelovy zákony o dědičnosti:
https://is.muni.cz/el/1441/jaro2010/Bi1BK_ZNT1/um/Mendelovy_zakony_o_dedicnosti.pdf
 - NZIP – Rejstříkový pojem:
<https://www.nzip.cz/rejstrikovy-pojem/>
 - Mendelovo muzeum – Gregor Johann Mendel:
<https://mendelmuseum.muni.cz/o-muzeu/gregor-johann-mendel>
- Správnost informací implementovaných do aplikace Biocross konzultoval **Mgr. Šimon Hrozinka**, který mi pomohl ověřit jejich korektnost a srozumitelnost.

Seznam obrázků

1	Ukázka kartičky	5
2	Ukázka stránky Pojmy	7
3	Příprava křížení	9
4	Výsledek křížení	10
5	Use Case pro stránku Classroom	11
6	Třívrstvá architektura	13
7	Schéma databáze	27
8	Deployment na stránce Render.com	29

Seznam ukázek kódu

1	Funkce pro míchání barev-mixColors(color1, color2)	17
2	Načtení hodnoty n-úhelníku	18
3	Inicializace canvasu a vymazání obsahu	18
4	Načtení barvy pro n-úhelník	18
5	Vykreslení n-úhelníku	18
6	Spuštění generátoru aplikace	20
7	Vytvoření kostry aplikace pomocí generátoru aplikace	20
8	Konfigurace v souboru package.json	21
9	Příkazy pro spuštění aplikace	21
10	Základní konfigurace aplikace v Express.js	21
11	Registrace routeru v app.js	22
12	Definice routeru v routes/classroom.js	22
13	Zpracování HTTP GET požadavku na /login	23
14	Zpracování POST požadavku v Express.js	23
15	Registrační formulář	24

16	Načítání kurzů na backendu	24
17	Načítání a zobrazení kurzů na frontendu	25
18	Instalace knihovny mysql2	27
19	Připojení k MySQL databázi	27
20	Načtení kurzů z databáze	28
21	Stažení aplikace	30
22	Přechod do složky projektu	30
23	Instalace závislostí	30
24	Spuštění aplikace	30