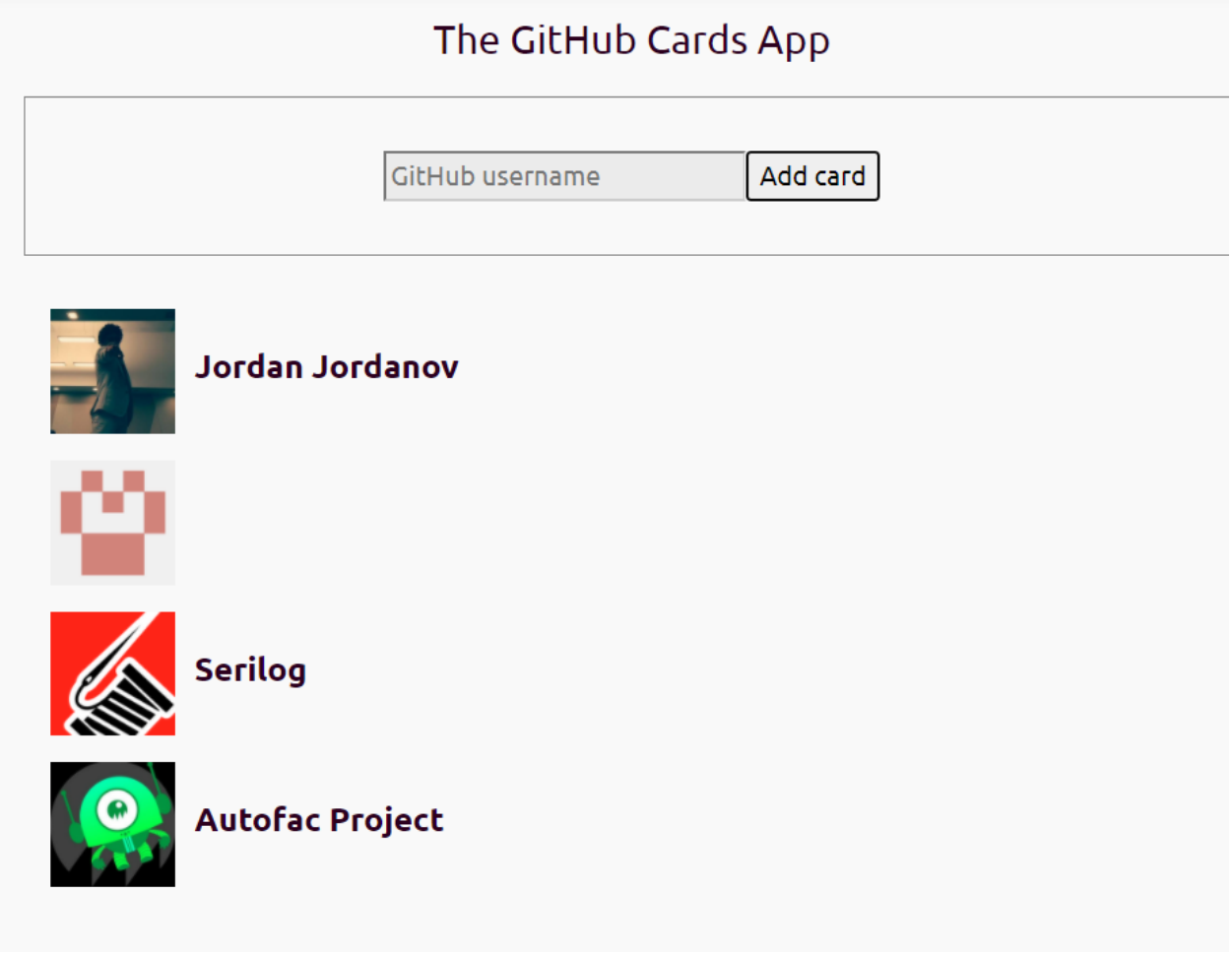
The GitHub Cards App



Какво ще изграждаме?

Виждали сме прости компоненти, работили сме с множество компоненти и сме виждали как и кога да използваме състоянието и свойствата на даден компонент. Все още обаче не сме работили с реални данни. Ще направим точно това в този модул на курса и ще използваме публичния сървиз на GitHub. Ще изградим прост компонент на карта с профили на GitHub, който показва информация за списъци с вашите профили. Ще има формуляр, в който потребителят може да въведе име от GitHub и да използва „Добавяне на карта“ за да добави нов към списъка с показаните профили. Ще се учим как да приемаме входни данни от потребителя и как да ги използваме, за да осъществите повиквания към GitHub API. Целта на това приложение е да ви осигури комфортна работа с обекти от данни. Другото нещо за това приложение е, че ще използваме класови компоненти. А след това по-голямото приложение ще бъде изцяло написано с функционални компоненти и „куки“.

React Class Components

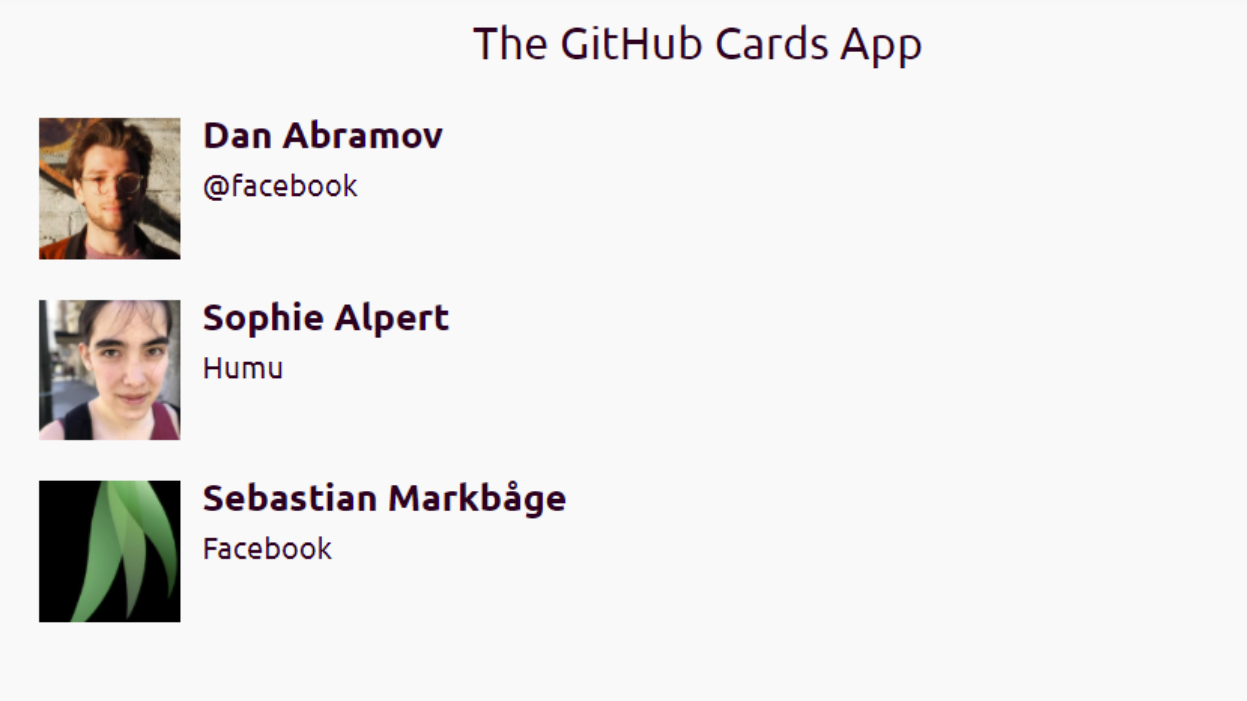
В http://jsdrops.com/rgs2.1 подготвих малък компонент на функцията на приложението и го прикачих към DOM. Ще използваме този, за да съдържа всички останали компоненти в това приложение. Първото решение, което трябва да вземете в приложение на React е структурата на компонентите. Трябва да решите колко компонента да използвате и какво трябва да описва всеки. Това често е лесно, ако имате пълната картина на приложението, което създавате, но на практика нямате. Нашето приложение ще бъде списък с карти от GitHub. Първата ви улика, че имате нужда от компонент, който да представлява една карта, и друг компонент, който да представлява самия списък Нека започнем с преобразуване на този компонент в клас. Вместо функционален компонент, дефинирате клас със същото име и наследявате специален клас в React, React.Component. Наследяването на React.Component прави класа ви официален компонент на React. Ще научим две основни концепции в този пример: конструктор и ключовата дума this в класове. Всеки React компонент трябва да има функция за рендиране. Това е единствената необходима функция. Тя връща виртуалното DOM описание на вашия компонент. Създаваме инстанции на класа и всяка получава своите свойства и състояние. Заглавието трябва да стане this.props.title. Създаваме класов компонент Card, който наследява React.Component, във функцията за рендиране връщаме div placeholder със клас github-профил.

Резултатът може да се провери на https://jscomplete.com/playground/rgs2.2 .

Работа с данни

Под връзката https://jscomplete.com/playground/rgs2.3 добавих временен масив testData, с който да работим. Тези данни бяха копирани от API на GitHub, така че след като приключим тестването на нашето приложение с тези фалшиви тестови данни, ще работи с реалните данни на API. URL адресът, който използвах за копиране на данните от GitHub, е api.github.com/users след това поставяте произволно потребителско име от GitHub и ще видите данните като JSON обект. Тъй като искаме да изобразим множество карти, се нуждаем от друг компонент, който да държи различните карти. Ще наречем този нов

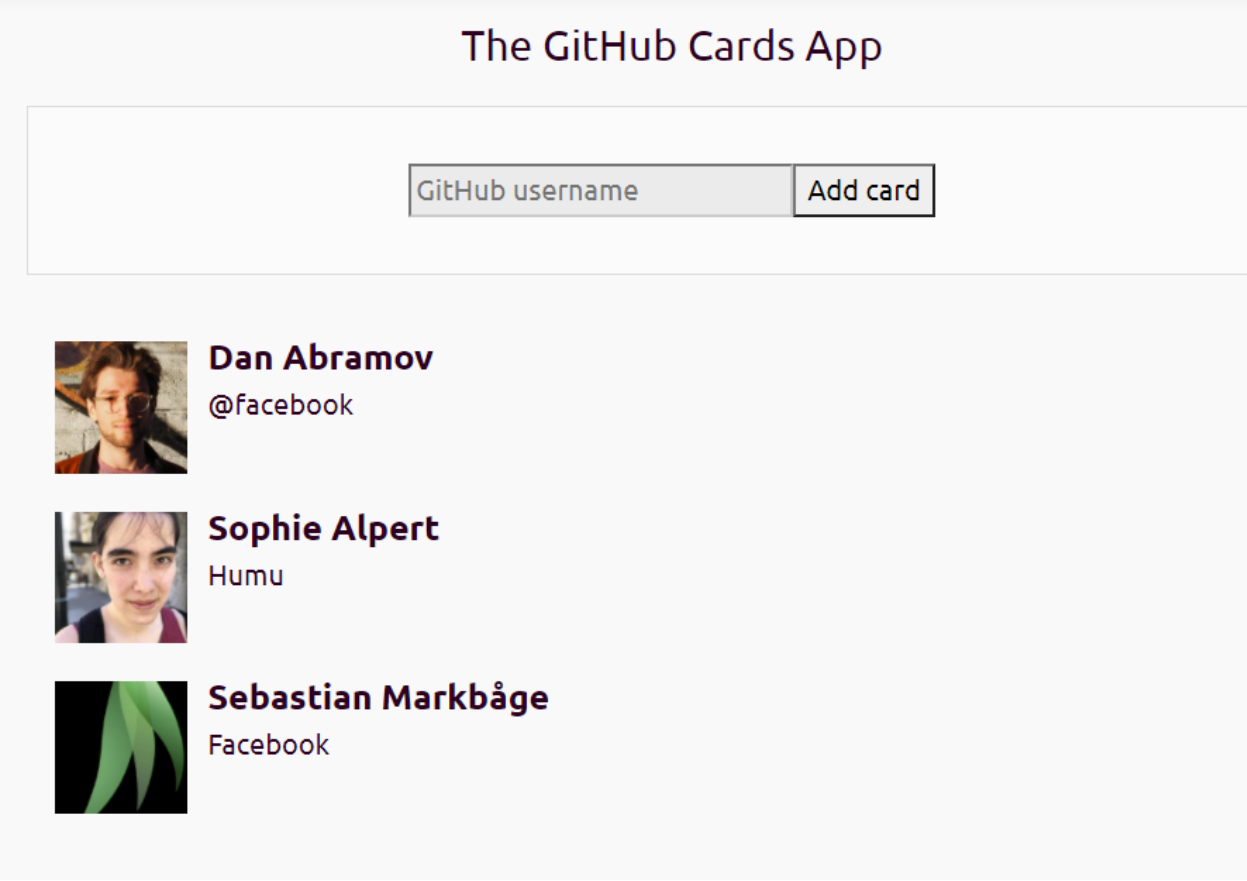
компонент CardList. Възможно е да управляваме състоянието на приложението в този компонент. Може да го създадем като функционален компонент. Функционалният компонент CardList ще получи стандартния аргумент за свойствата връща div, който ще съдържа нашия списък с карти. В този div ще изобразим компонент Card и ще трябва да променим компонента на приложението да изобрази CardList вместо Card. Нека поставим някои реални данни в компонента Card. За това ще използваме масива testData. Вътре в функцията за рендиране на компонента Card, ще създадем локална променлива. За изображението ще използваме profile.avatar_url а за името се нуждаем от profile.name. А за името на компанията се нуждаем от profile.company. Компонента Card ще получава данните чрез свойствата. Един от начините да направите това е да вземете обекта, който държи свойствата и да го разнесете вътре в елемента на компонента Card. РЕЗУЛТАТ: https://jscomplete.com/playground/rgs2.4 .



Инициализиране и прочитане обекта на състояние (State Object)

За да вземем евентуално въвеждане от потребителя, можем да използваме прост HTML формуляр с вход и бутон. Нека създадем нов React компонент: Form, който наследява React.Component. Той се нуждае от функция за рендиране и трябва да върне някакъв DOM. Използваме входен елемент за да му дадем placeholder + бутон за добавяне на нова карта. За да се покаже този компонент в браузъра, трябва да го включим някъде в това, което изобразяваме. Подходящо място на компонента Form е в компонента на приложението на най-високо ниво. Компонентът App ще обработва връзката между компонента CardList и

компонента Form. Масивът от данни, движещ кода, който имаме до момента, все още е глобална променлива. Компонентите трябва да избягват четенето на глобални променливи. Вместо да го четем директно от компонента CardList, нека го направим част от компонента App и да накараме компонента CardList да го получи като свойтсво. Сега в компонента CardList, вместо testData директно, трябва да използваме props.profiles. Компонентът Form ще трябва да добави запис към този масив. За да позволим както CardList, така и компонентът Form да имат достъп до масива на профилите, трябва да го поставим в състоянието на самия компонент на приложението. За да инициализираме обект на състояние за компонента App, трябва да добавим конструктор. Този специален метод получава и свойствата на инстанцията. Трябва да извика JavaScript супер метода, който е необходим за зачитане на връзката между класа на приложението и класа, от който се простира React.Component. React ще се оплаче, ако не го направите. Очакван резултат: https://jscomplete.com/playground/rgs2.5



Вземане на данни от потребителя

За да вземем данни от потребителя, трябва да дефинираме манипулатор на събития в React UI. На бутона „Добавяне на карта“ можем да дефинираме събитие onClick, но предпочитам да го обработим със събитие onSubmit за самия елемент на HTML формуляр. Използвайки onSubmit, можете да използвате функциите за подаване на естествени формуляри. Например можете да направите това въвеждане задължително и събитието onSubmit ще отчете това в браузъра. Нека дефинираме нова функция handleSubmit**.** Трябва да предотвратим поведението по подразбиране за подаване на формуляр, като използваме event.preventDefault. Това е важно, когато работите с формуляри, защото ако изпратите формуляра, страницата ви ще се обнови. Нуждаем се от събитие onChange, за да може DOM да каже на React, че нещо се е променило в този вход и трябва да го отразите и в потребителския интерфейс. Ще използваме вградена функция за събитието onChange. Тази функция получава обекта на събитието като свой аргумент и трябва да промени стойността на елемента на състоянието userName в това, което е въведено в текстовото поле. За целта в компонента използваме this.setState и му предаваме обект, който има новото състояние. В този случай трябва да го предадем на username като вземете стойността, която потребителят е въвел директно от DOM, като използваме event.target.value. Това е синтаксисът, който трябва да използваме, за да променим състоянието на класов компонент. Резултатът може да видите тук: https://jscomplete.com/playground/rgs2.6

```
class Form extends React.Component {
  state = { userName: '' };
  handleSubmit = (event) => {
    event.preventDefault();
    console.log(this.state.userName);
  };
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <input
          type="text"
          value={this.state.userName}
          onChange={event => this.setState({ userName: event.target.value })}
          placeholder="GitHub username"
          required
        />
        <button>Add card</button>
      </form>
    );
  }
}
```

AJAX

Готови сме да поискаме от GitHub API данни за профила. В този инструмент имаме вградена axios библиотека. Във функцията handleSubmit трябва да направим: axios.get и след това да посочим крайната точка на APIто, от където са данните. Ще инжектираме стойността на потребителското име, което потребителят е въвел в полето или this.state.userName. Axios.get връща обещание (promise, което можем да изчакаме и то ще ни върне обект, като отговор. Трябва да маркираме handleSubmit като асинхронен. Въведете всяко валидно потребителско име за GitHub, щракнете върху „Добавяне на карта“ и handleSubmit ще отиде и ще извлече данните от профила на конкретното потребителско име и ще отпечата отговора. Този отговор има поле data, което съдържа действителната информация за профила от GitHub. (response.data). AXIOS: връща обект като отговор (JSON данни, анализирани и готови за нас). Дефинираме нова функция на компонента App: addNewProfile - тази функция ще получи profileData като аргумент и ще актуализира състоянието, след първоначалната настройка. Предаваме на компонента Form ново свойсвтво: onSubmit със стойност функцията: this.addNewProfile.В рамките на функцията handleSubmit можем да получим достъп до новото свойство this.props.onSubmit. Тази препратка към функция е псевдоним на функцията addNewProfile в компонента App. И като аргумент предаваме атрибута data, идващ от AJAX отговора. Поставяме profileData в състоянието на компонента App. Трябва да добавим този обект към масива, които сме съхранили като профили в състоянието. И за да го направим извикаме setState. Тази функция ще ви даде достъп до предишното състояние. И това, което върнете тук, се превръща в новото състояние. Прочитаме с помощта на prevState.profiles и след това добавяме новия profileData. Можем да премахнем всички тестови данни и да стартираме масива от профили като празен. Едно малко подобрение е да нулираме полето за потребителско име, след като приключим с добавянето му към масива с профили. Данните на GitHub се доставят с уникален идентификатор.
Има много подобрения, които можем да направим в този код. Трябва да обработим грешки. Какво трябва да направи потребителският интерфейс, ако потребителят въведе невалиден GitHub? Какво трябва да направи потребителският интерфейс, ако заявката за данни се провали в мрежата? Какво трябва да направи потребителският интерфейс, ако заявката отнема твърде много време? Цялото ви приложение всъщност не трябва да зависи пряко от библиотека като axios например. Трябва да имате малък модул от типа агент, който има една отговорност да комуникира с външни API и да прави вашият код да зависи само от този агент. Разгледайте крайния код: http://jsdrops.com/rgs2.7 . Едно бързо упражнение, което можете да направите, да преобразувате всички класови компоненти във функционални компоненти. (вместо състояние ще използвате „кука“ и вместо this.setState използвате функцията за актуализиране) .

Обобщение

Изградихме няколко прости React компонента:

1) Card, за да визуализира информация за профил в GitHub,

2) CardList, за да преобразува масив от записи в масив от компоненти на Card

3) компонент форма за четене на данни от потребителя

4) компонент на приложение за управление на връзката между всички

Създаване на локална среда за разработка

Създаването на среда за разработка не е сред най-забавните неща. Трябва да накарате много различни инструменти да работят заедно. Инструментите имат различни APIта и всеки инструмент ще трябва да бъде конфигуриран. Възможно е да срещнете проблеми като различни версии да не са съвместими и да не работят заедно. Цялата среда може внезапно да спре да работи след определено надграждане на инструментите и ще трябва да отделите известно време за отстраняване на грешки. Освен това средата за разработка се различава от продуктивната среда, което означава, че това, което работи за вас в разработката, може да не работи в производството. За щастие има някои инструменти на високо ниво, които можете да използвате, за да избегнете някои от кошмарите за работа със среди.

Vite

Проектът Vite може да ви помогне да създадете локална среда за разработка на React с помощта на една команда.

npm install vite@latest

Един от начините за стартиране на ново приложение е:

npm create vite@latest

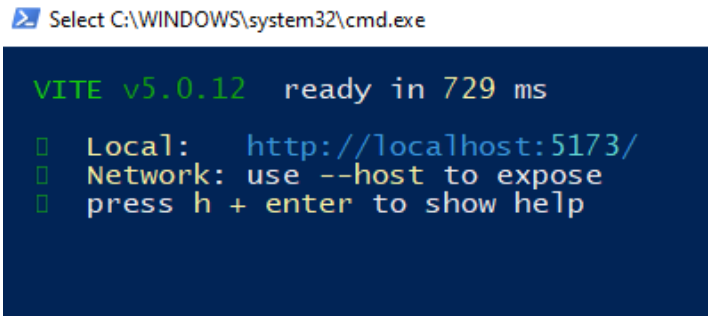
√ Project name: gitcards
√ Select a framework: » React
√ Select a variant: » JavaScript

Очакван резултат:
Scaffolding project in ...\gitcards

След това изпълнете:

- cd gitcards
- npm install
- npm run dev

Инсталацията на пакетите ще отнеме около 2/3 минути.



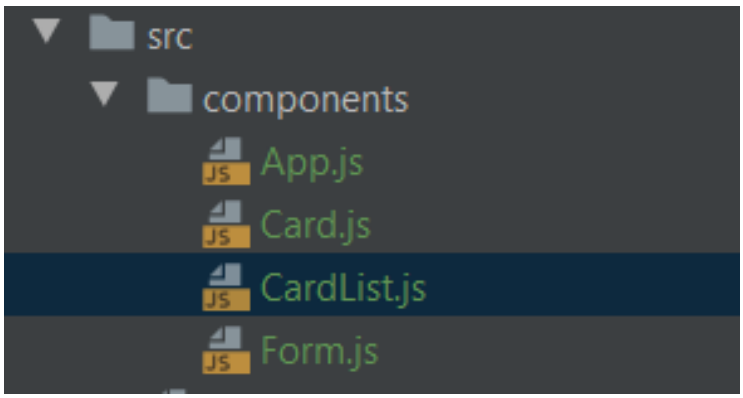
Прекъсваме програмата (Ctrl + C), за да инсталираме библиотеката Axios, чрез командата:

npm add axios

Можем да отворим **Visual Studio Code** с командата **code .** ; След това, в директорията **src** добавете нова (папка) с наименование **components,** след това преместете **App.jsx** в нея и изтривате **assets** и **App.css**.

Добавете 3 нови **.jsx** фаила в **components**:

- Card.jsx
  Добавете съдържанието от http://jsdrops.com/rgs2.7 , като добавите **import * as React from "react";** и **export default Card;**
- CardList.jsx
  Този компонент ще има нужда от **import Card from "./Card";**
- Form.jsx
  А тук трябва да добавим:
  import * as React from "react";
  import axios from 'axios';
- App.jsx
  Променете този компонент , като използвате ресурса от по горния линк и добавите 2та по горни компонента.



Променете пътя до **App**.js в **index.js** по следния начин: *import App from './components/App';*

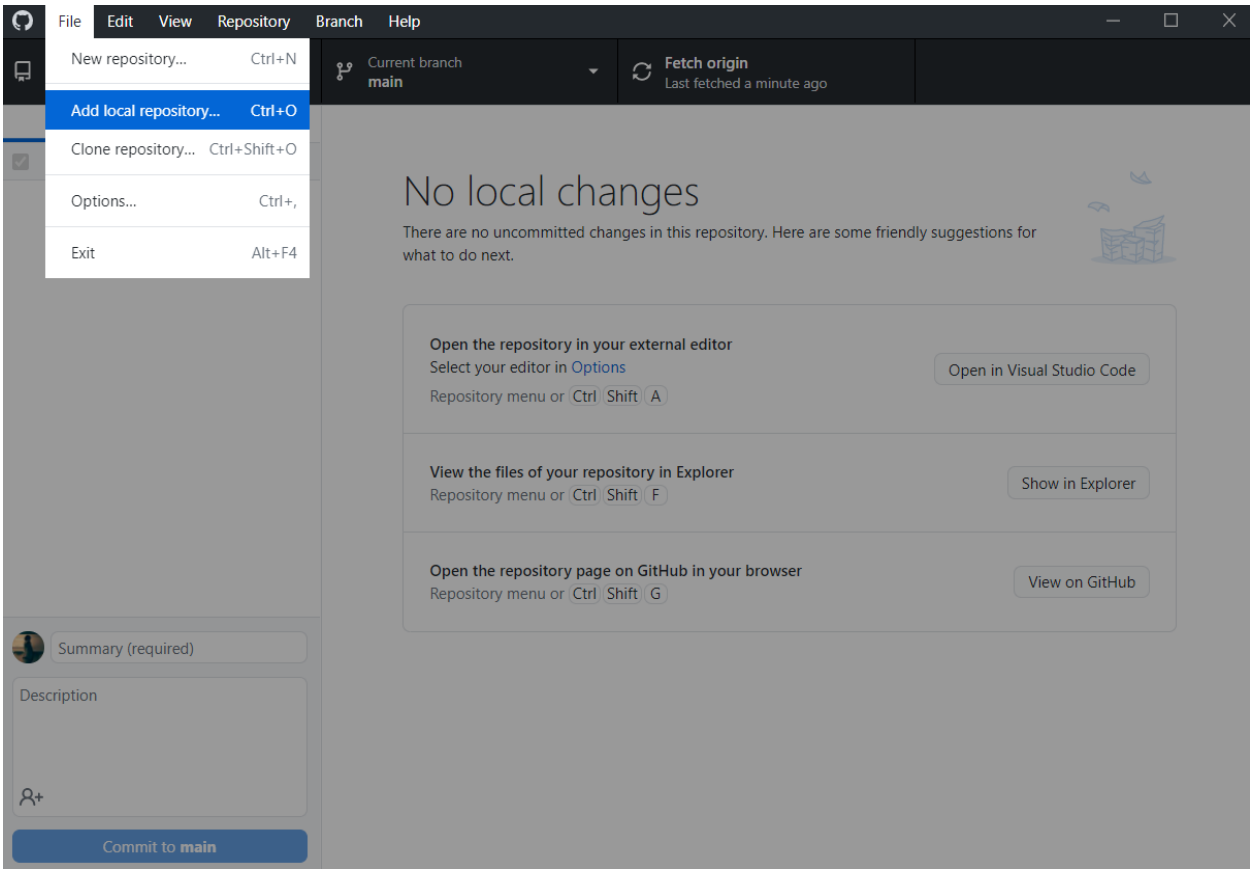
Изпълнете **npm run dev** и се уверете, че всичко работи, както е очаквано**.**

Бонус: Заменете CSS, който е по подразбиране, с Bootstrap:

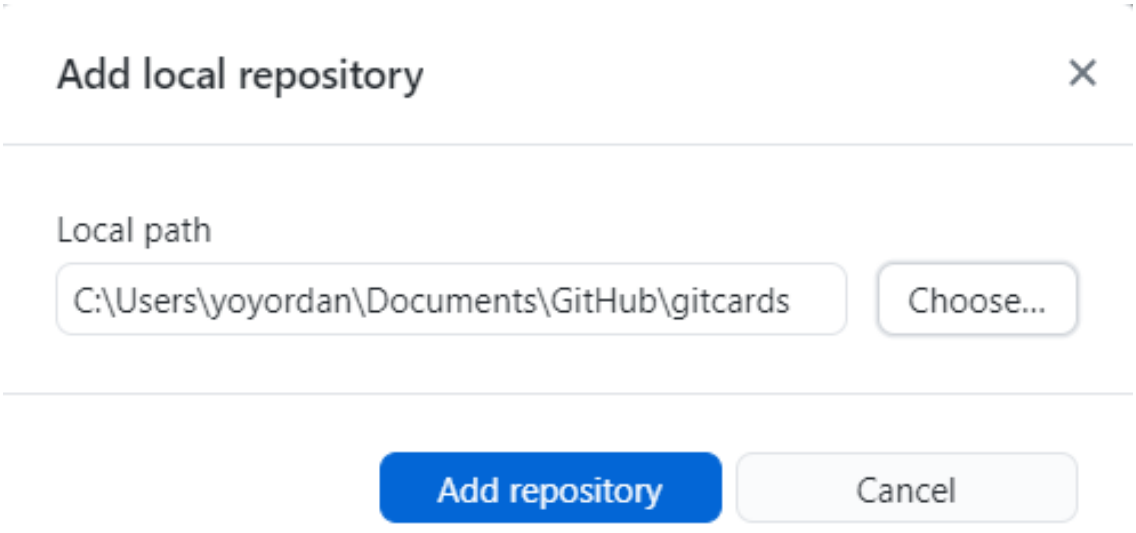
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossOrigin="anonymous" />

Как да добавим в GitHub новосъздадения ни проект:

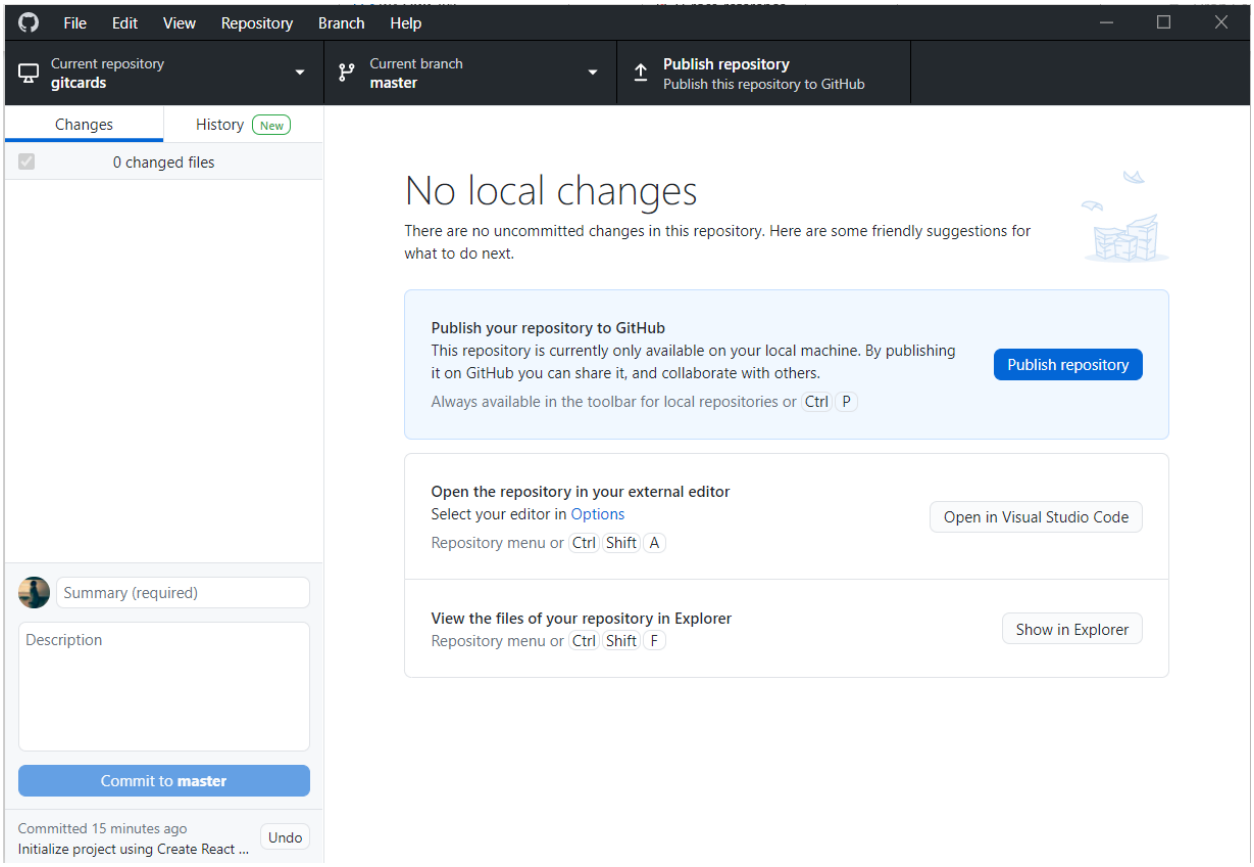
1) Отворете GitHub Desktop , натиснете върху File => Add local repository



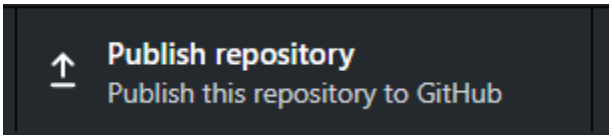
2) Навигирайте до директорията на приложението



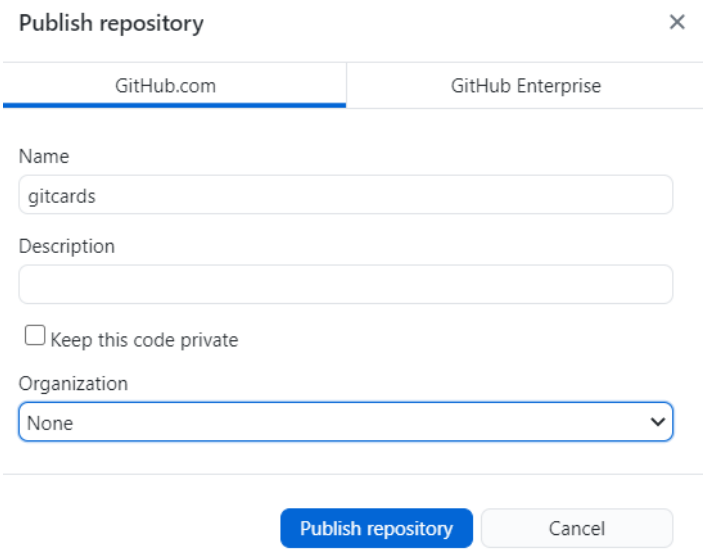
3) Текущото положение трябва да е:



4) Публикувайте хранилището, като натиснете върху Publish repository



5) Настройте кода да бъде публичен, като махнете отметката от **Keep this code private**



ДОМАШНА РАБОТА

Създайте приложение, подобно на „GitHub Cards“, като използвате API-то на мерinджей:

https://students-manager.azurewebsites.net/api/students/

До това API се осъществява достъп чрез Basic Authentication със следните данни:

• Потребителско име: "guest"

• Парола: "guest"

Основната разлика спрямо api.github.com е, че Student Manager API предоставя колекция от записи, а не единичен обект. Затова при стартиране на приложението трябва да изтеглите всички записи, а при търсене — да филтрирате вече заредените в паметта обекти.