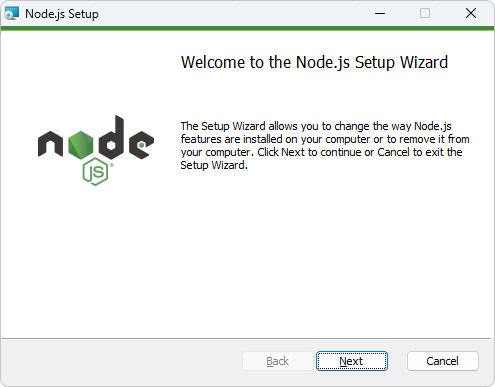
Node.js 官方 下载地址

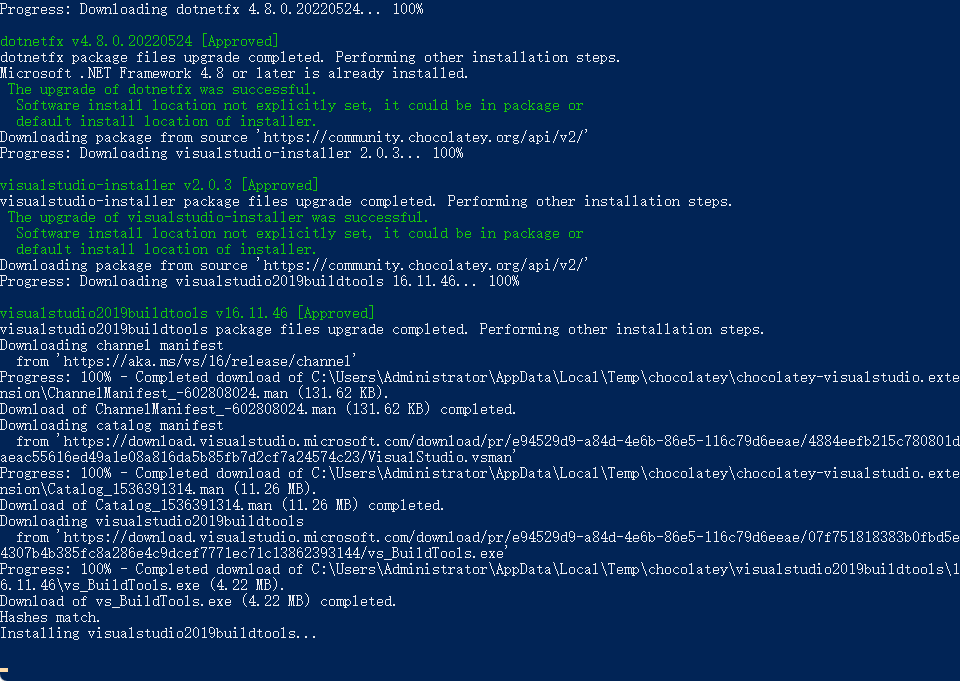
<https://nodejs.org/zh-cn/download>

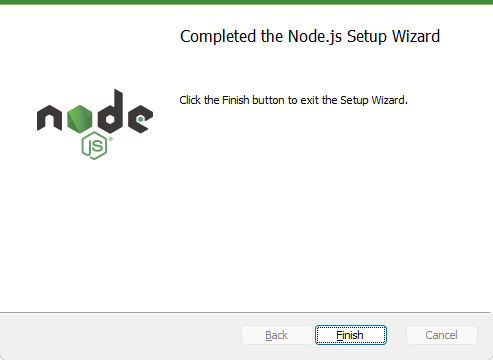


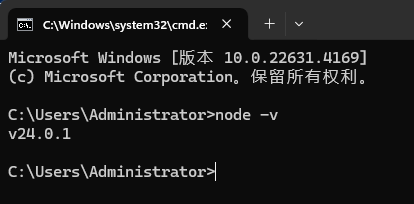




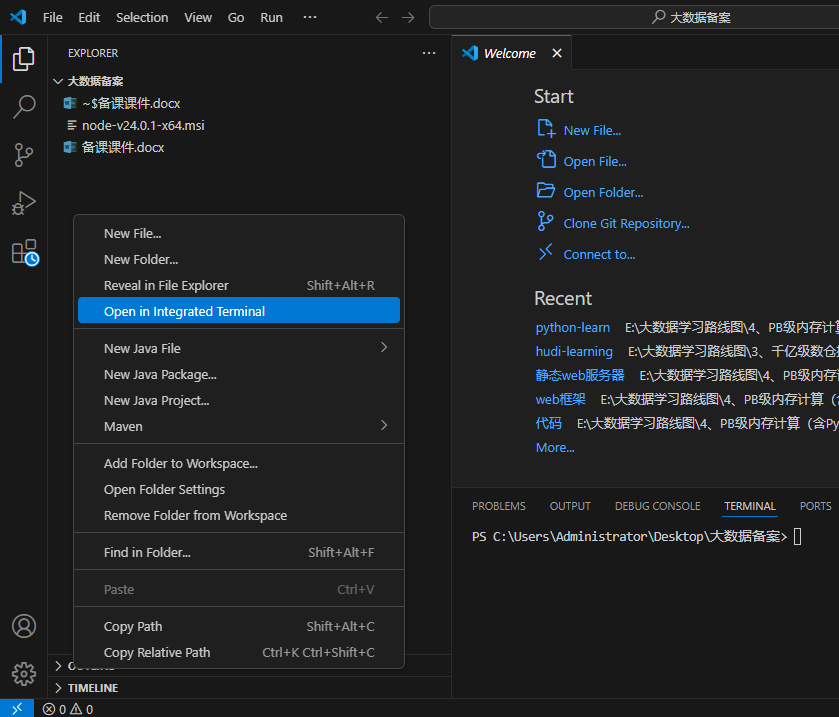
**默认安装地址，一直下一步即可**







 以下vsCode编辑器的下载地址 <https://code.visualstudio.com/>



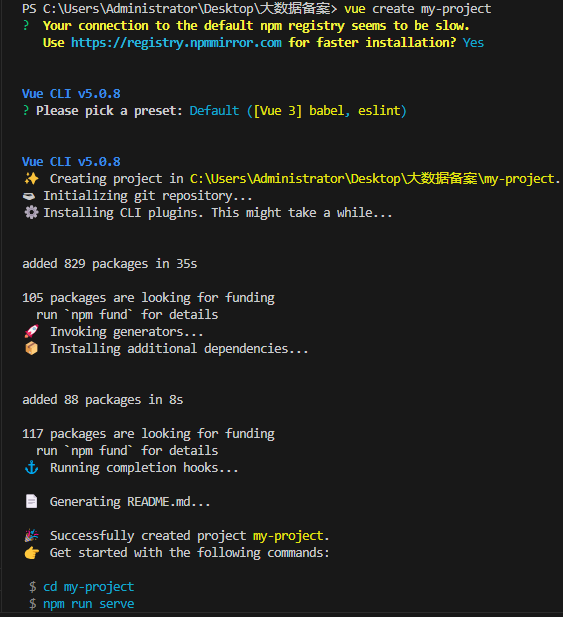
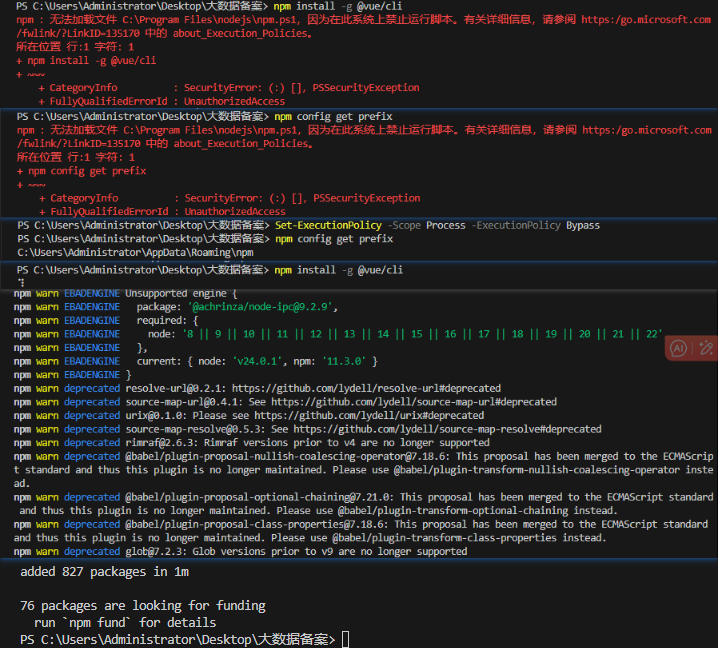
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser

npm config get prefix

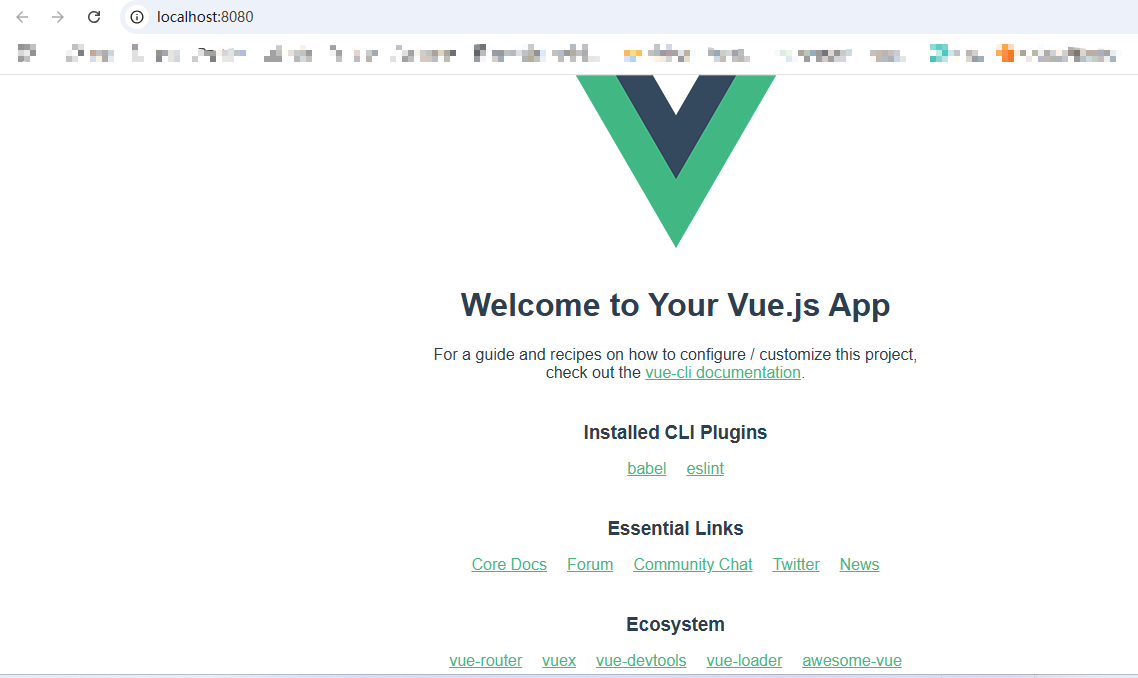
npm install -g @vue/cli

vue create my-project

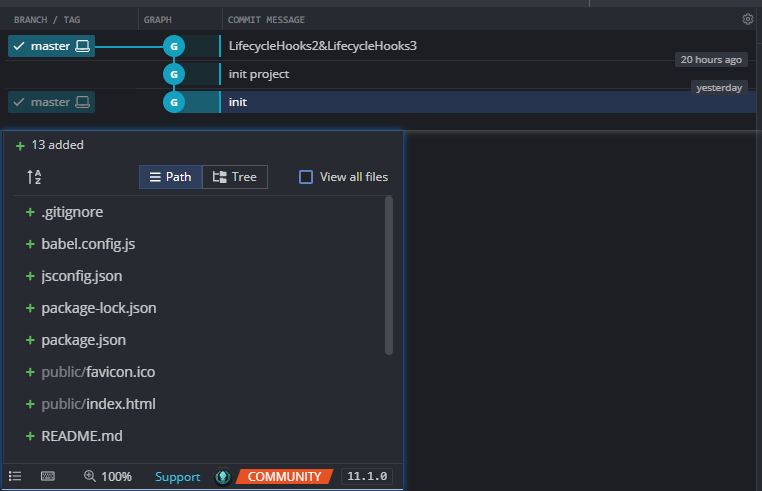


npm run serve





 以下GitKraken 代码版本控制的管理工具下载地址 <https://www.gitkraken.com/>

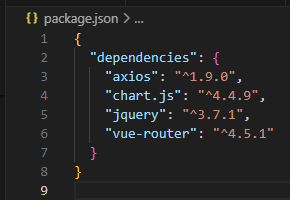


npm install jquery --save

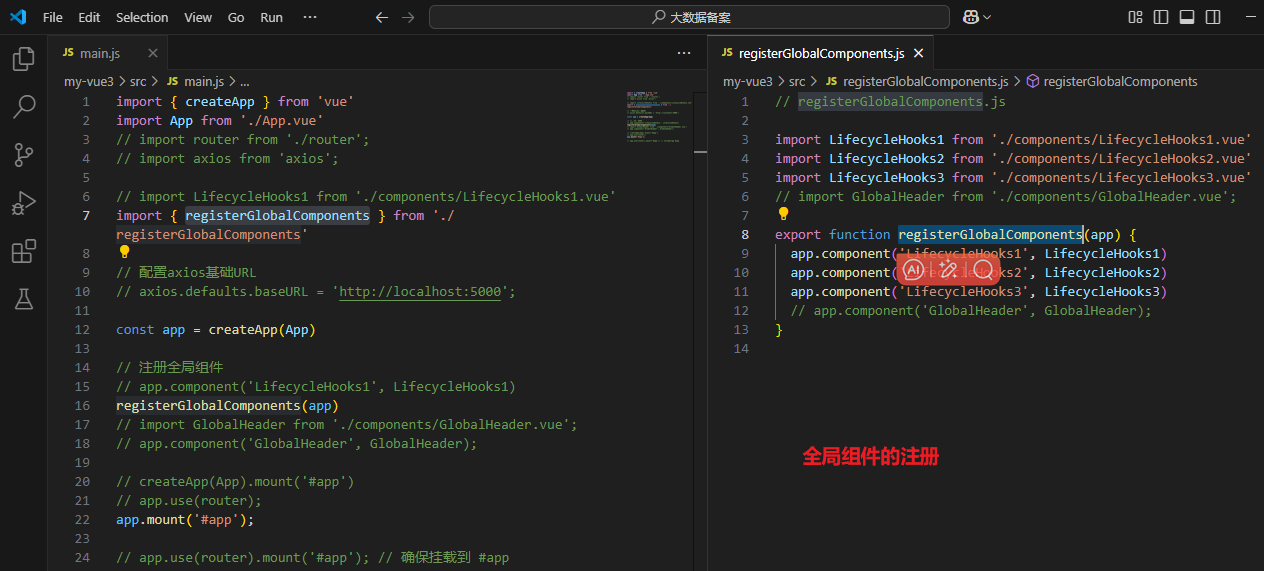
npm install axios --save

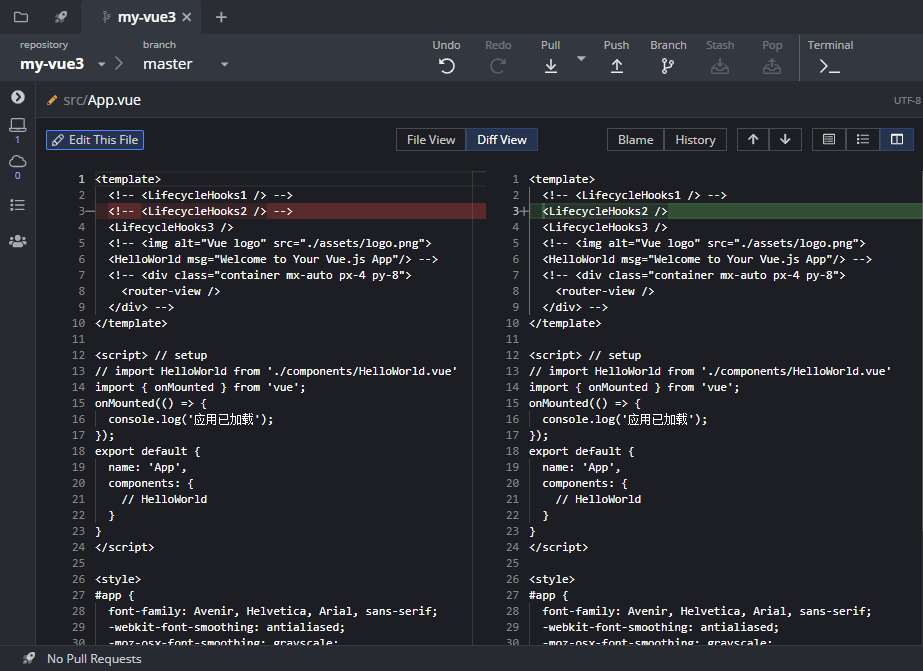
npm install chart.js --save

npm install vue-router --save



以下的第一部分的 **基于Vue.js 3的组件注册与引用** 课件小结





**全局组件的引用 和 局部组件的注册与引用**



**以上提供github下载地址**： <https://github.com/janusit/testVue3.git>

以下的第二部分的 **基于Vue.js 3的组件注册与引用** 课件小结

### **教学方案说明**

这个教学平台方案包含以下核心内容：

****组件系统****

* 1. 全局组件：GlobalHeader（自动注册）
  2. 局部组件：QueryForm、ResultTable
  3. 动态组件：根据路由加载的QueryView和AnalysisView

****数据查询功能****

* 1. 表单组件支持选择表名、字段和查询条件
  2. 通过 axios 与 Python API 通信
  3. 结果表格组件展示查询结果

****数据分析功能****

* 1. 支持多种统计分析类型（计数、平均、总和等）
  2. 使用 Chart.js 实现数据可视化
  3. 动态生成图表展示分析结果

****后端 API****

* 1. 使用 FastAPI 构建 RESTful API
  2. 连接 MySQL 数据库执行查询
  3. 处理数据分析请求并返回结果

### **教学实施建议**

****环境搭建****

* 1. 先安装 Vue 3 和相关依赖
  2. 配置 Python 环境和 FastAPI
  3. 准备 MySQL 数据库和测试数据

****教学步骤****

* 1. 讲解 Vue 3 组件注册与引用机制
  2. 演示如何创建表单组件和数据展示组件
  3. 讲解前后端数据交互流程
  4. 实现数据分析功能和可视化展示

****扩展练习****

* 1. 添加更多数据分析功能
  2. 实现数据导出功能
  3. 优化 UI 设计和用户体验
  4. 添加用户认证和权限管理

┌───────────────────────────────────────────────────────────┐

│ 前端部分 │

│ ┌─────────────┐ ┌─────────────┐ ┌───────────────────┐ │

│ │ 组件系统 │ │ 数据可视化 │ │ 用户交互界面 │ │

│ │ - 全局组件 │ │ - 图表组件 │ │ - 登录页面 │ │

│ │ - 局部组件 │ │ - 表格组件 │ │ - 数据查询页面 │ │

│ │ - 动态组件 │ │ - 统计卡片 │ │ - 分析结果页面 │ │

│ └─────────────┘ └─────────────┘ └───────────────────┘ │

│ Vue 3 │

└─────────────────────────────┬───────────────────────────────┘

│

▼

┌───────────────────────────────────────────────────────────┐

│ 后端部分 │

│ ┌─────────────┐ ┌─────────────┐ ┌───────────────────┐ │

│ │ Python API │ │ 数据库访问 │ │ 数据分析处理 │ │

│ │ - FastAPI │ │ - MySQL │ │ - Pandas/Numpy │ │

│ │ - 路由管理 │ │ - 查询构建 │ │ - 统计计算 │ │

│ └─────────────┘ └─────────────┘ └───────────────────┘ │

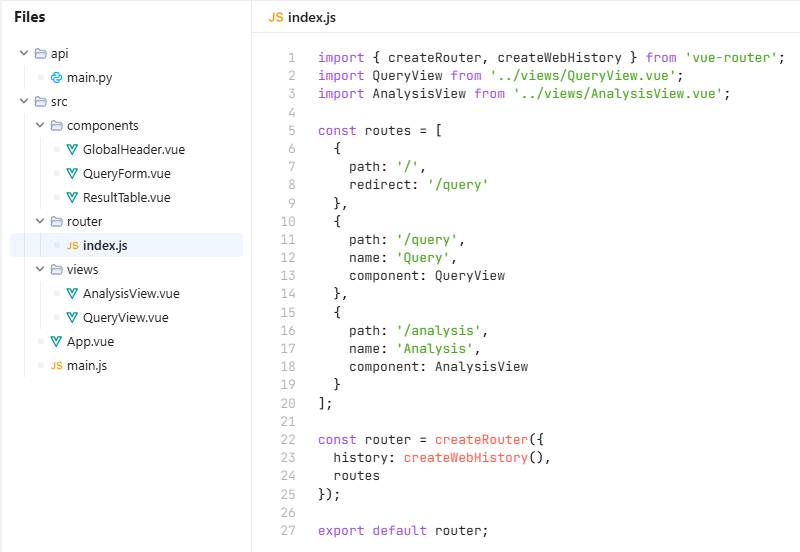
└───────────────────────────────────────────────────────────┘

### pip install mysql-connector-python

pip list | grep mysql-connector-python

where python

需要修改第一部分课件的git代码的三个文件，下图是说明



提供接口服务的示例代码，需要python + mysql, 本例子是在虚拟机中 构建的 数据库数据，以下是 main.py

from fastapi import FastAPI, HTTPException

from pydantic import BaseModel

import pandas as pd

import mysql.connector

from mysql.connector import Error

app = FastAPI()

# 数据库配置

db\_config = {

    'host': 'hudi',

    'database': 'db\_msg',

    'user': 'root',

    'password': '123456'

}

class QueryRequest(BaseModel):

    table: str

    fields: str

    condition: str

class AnalysisRequest(BaseModel):

    type: str

    field: str

@app.post("/api/query")

async def execute\_query(query: QueryRequest):

    try:

        connection = mysql.connector.connect(\*\*db\_config)

        if connection.is\_connected():

            cursor = connection.cursor(dictionary=True)

            query\_str = f"SELECT {query.fields} FROM {query.table}"

            if query.condition:

                query\_str += f" WHERE {query.condition}"

            cursor.execute(query\_str)

            records = cursor.fetchall()

            return records

    except Error as e:

        raise HTTPException(status\_code=500, detail=str(e))

    finally:

        if connection.is\_connected():

            cursor.close()

            connection.close()

@app.post("/api/analyze")

async def run\_analysis(analysis: AnalysisRequest):

    try:

        connection = mysql.connector.connect(\*\*db\_config)

        if connection.is\_connected():

            cursor = connection.cursor(dictionary=True)

            # 根据分析类型执行不同的SQL查询

            if analysis.type == 'count':

                query\_str = f"SELECT COUNT({analysis.field}) as result FROM users"

            elif analysis.type == 'avg':

                query\_str = f"SELECT AVG({analysis.field}) as result FROM users"

            elif analysis.type == 'sum':

                query\_str = f"SELECT SUM({analysis.field}) as result FROM users"

            elif analysis.type == 'max':

                query\_str = f"SELECT MAX({analysis.field}) as result FROM users"

            elif analysis.type == 'min':

                query\_str = f"SELECT MIN({analysis.field}) as result FROM users"

            else:

                raise HTTPException(status\_code=400, detail="不支持的分析类型")

            cursor.execute(query\_str)

            result = cursor.fetchone()

            # 获取图表数据

            chart\_query = "SELECT MONTH(created\_at) as month, COUNT(\*) as count FROM users GROUP BY MONTH(created\_at)"

            cursor.execute(chart\_query)

            chart\_data = cursor.fetchall()

            labels = [f"{row['month']}月" for row in chart\_data]

            values = [row['count'] for row in chart\_data]

            return {

                'result': result['result'],

                'chartData': {

                    'labels': labels,

                    'values': values

                }

            }

    except Error as e:

        raise HTTPException(status\_code=500, detail=str(e))

    finally:

        if connection.is\_connected():

            cursor.close()

            connection.close()

如需后续详情...敬请期待...