```
class Database {
public:
    Database(const Database&) = delete;
    static void addTest(std::string title, std::vector<Entry> entries);
    static uint32_t getNextEntryID();
    static uint32_t getNextTestID();
private:
    Database();
    static Database instance;
    uint32_t nextTestID = 0;
    uint32_t nextEntryID = 0;
    std::vector<TestStruct> tests;
};
```

W swoim projekcie zastosowałem wzorzec Singleton. Działa on w taki sposób, że istnieje tylko jedna instancja danej klasy. W przypadku naszego projektu niepotrzebnym jest (a wręcz jest niepożądane) tworzenie kilku obiektów bazy danych. Usunięcie konstruktora kopiującego oraz ustawienie konstruktora domyślnego jako prywatnego uniemożliwia tworzenie kolejnych instancji bazy. Jedna, utworzona instancja znajduje się wewnątrz samej klasy.

```
#include "Database.h"

Database Database::instance;

Database::Database() {}

uint32_t Database::getNextEntryID() {
    return instance.nextEntryID++;
}

uint32_t Database::getNextTestID() {
    return instance.nextTestID++;
}

void Database::addTest(std::string title, std::vector<Entry> entries) {
    TestStruct test;
    test.ID = getNextTestID();
    test.title = title;
    test.entries = entries;
    test.started = false;
    instance.tests.push_back(test);
}
```

Utworzenie instancji odbywa się w pliku klasy z rozszerzeniem .cpp. Ważnym jest, aby zdefiniować konstruktor domyślny – nawet, jeśli nic nie wykonuje. W przeciwnym wypadku kompilator zwróci błąd. Praca z polami instancji odbywa się poprzez metody, które edytują instancję (nie samą klasę, stąd np. `return instance.nextEntryID++` zamiast `nextEntryID++`);

Dzięki temu rozwiązaniu odwołanie do Singletona może odbyć się z każdego pliku źródłowego przy użyciu zapisu np. `Database::getNextEntryID()`. Nie ma konieczności tworzenia obiektu bazy danych.