### Create Architecture for ola

### 1. ****Planning****

* **Objective**: Define what the app needs to do—like allowing passengers to book rides and drivers to accept them.
* **Scope**: Outline features such as user accounts, ride tracking, payment processing, and notifications.

### 2. ****Analysis****

* **Requirements Gathering**: Understand what users need and how the system should work. For example, passengers need to request rides, and drivers need to accept them.
* **System Design**: Decide on how the app will be structured. This includes the user interfaces, server components, and databases.

### 3. ****Design****

* **User Interfaces**: Create designs for the passenger and driver apps. This includes screens for booking rides, tracking locations, and managing profiles.
* **Backend Architecture**: Plan how the server will handle tasks:
  + **User Management**: Handles sign-ups, logins, and profile updates.
  + **Ride Management**: Manages ride requests, assignments, and status updates.
  + **Payment Processing**: Manages how payments are processed and recorded.
  + **Notifications**: Sends updates and alerts to users.
  + **Analytics**: Tracks data on app usage to improve services.

### 4. ****Implementation****

* **Coding**: Develop the passenger and driver apps, and build the server-side components based on the design.
  + **Frontend Development**: Build the mobile or web apps that users interact with.
  + **Backend Development**: Create the server-side logic and APIs that process requests and manage data.

### 5. ****Testing****

* **Unit Testing**: Test individual parts of the app to ensure they work correctly.
* **Integration Testing**: Test how different parts of the app work together.
* **User Acceptance Testing**: Have real users test the app to make sure it meets their needs and is easy to use.

### 6. ****Deployment****

* **Release**: Publish the app to app stores or make it available for users to download.
* **Monitoring**: Keep an eye on the app’s performance and fix any issues that come up.

### 7. ****Maintenance****

* **Updates**: Regularly update the app to add new features, fix bugs, and improve performance.
* **Support**: Provide help to users and address any issues they encounter.

**Example 2:**

#include <iostream>

template <typename T>

class DynamicArray {

private:

T\* array;

size\_t capacity;

size\_t size;

void resize(size\_t new\_capacity) {

T\* new\_array = new T[new\_capacity];

for (size\_t i = 0; i < size; i++) {

new\_array[i] = array[i];

}

delete[] array;

array = new\_array;

capacity = new\_capacity;

}

public:

DynamicArray() : capacity(1), size(0) {

array = new T[capacity];

}

~DynamicArray() {

delete[] array;

}

void insert(const T& value) {

if (size == capacity) {

resize(capacity \* 2);

}

array[size++] = value;

}

void remove(size\_t index) {

if (index >= size) {

std::cerr << "Index out of bounds" << std::endl;

return;

}

for (size\_t i = index; i < size - 1; i++) {

array[i] = array[i + 1];

}

size--;

if (size > 0 && size == capacity / 4) {

resize(capacity / 2);

}

}

void clear() {

delete[] array;

array = new T[capacity = 1];

size = 0;

}

size\_t getSize() const {

return size;

}

size\_t getCapacity() const {

return capacity;

}

T& operator[](size\_t index) {

if (index >= size) {

throw std::out\_of\_range("Index out of bounds");

}

return array[index];

}

const T& operator[](size\_t index) const {

if (index >= size) {

throw std::out\_of\_range("Index out of bounds");

}

return array[index];

}

};

int main() {

DynamicArray<int> arr;

arr.insert(10);

arr.insert(20);

arr.insert(30);

std::cout << "Array contents: ";

for (size\_t i = 0; i < arr.getSize(); i++) {

std::cout << arr[i] << " ";

}

std::cout << std::endl;

arr.remove(1);

std::cout << "After removal: ";

for (size\_t i = 0; i < arr.getSize(); i++) {

std::cout << arr[i] << " ";

}

std::cout << std::endl;

arr.clear();

std::cout << "After clearing, size: " << arr.getSize() << std::endl;

return 0;

}

**Example 3:**

#include <iostream>

#include <stdexcept>

#include <string>

template <typename T>

class Stack {

private:

T\* array;

size\_t capacity;

size\_t topIndex;

void resize(size\_t new\_capacity) {

T\* new\_array = new T[new\_capacity];

for (size\_t i = 0; i < topIndex; i++) {

new\_array[i] = array[i];

}

delete[] array;

array = new\_array;

capacity = new\_capacity;

}

public:

Stack() : capacity(10), topIndex(0) {

array = new T[capacity];

}

~Stack() {

delete[] array;

}

void push(const T& value) {

if (topIndex == capacity) {

resize(capacity \* 2);

}

array[topIndex++] = value;

}

void pop() {

if (topIndex == 0) {

throw std::out\_of\_range("Stack underflow");

}

topIndex--;

}

T& peek() {

if (topIndex == 0) {

throw std::out\_of\_range("Stack is empty");

}

return array[topIndex - 1];

}

bool isEmpty() const {

return topIndex == 0;

}

size\_t size() const {

return topIndex;

}

};

int main() {

Stack<int> intStack;

Stack<float> floatStack;

Stack<std::string> stringStack;

// Testing with int stack

intStack.push(10);

intStack.push(20);

std::cout << "Top of int stack: " << intStack.peek() << std::endl;

intStack.pop();

std::cout << "Top of int stack after pop: " << intStack.peek() << std::endl;

// Testing with float stack

floatStack.push(1.5f);

floatStack.push(2.5f);

std::cout << "Top of float stack: " << floatStack.peek() << std::endl;

// Testing with string stack

stringStack.push("Hello");

stringStack.push("World");

std::cout << "Top of string stack: " << stringStack.peek() << std::endl;

stringStack.pop();

std::cout << "Top of string stack after pop: " << stringStack.peek() << std::endl;

return 0;

}

**Example 4:**

#include <iostream>

#include <fstream>

#include <string>

#include <stdexcept>

void readFile(const std::string& filename) {

std::ifstream file;

file.open(filename);

if (!file) {

throw std::runtime\_error("File not found: " + filename);

}

std::string line;

while (std::getline(file, line)) {

try {

if (line.empty()) {

throw std::runtime\_error("Empty line encountered");

}

std::cout << "Read line: " << line << std::endl;

} catch (const std::exception& e) {

std::cerr << "Error reading line: " << e.what() << std::endl;

}

}

if (file.bad()) {

throw std::runtime\_error("Error while reading the file");

}

file.close();

}

int main() {

std::string filename;

std::cout << "Enter the filename: ";

std::cin >> filename;

try {

readFile(filename);

} catch (const std::exception& e) {

std::cerr << "An error occurred: " << e.what() << std::endl;

}

return 0;

}

**Example 5:**

Google Test Unit Test Suite

// test\_DynamicArray.cpp

#include <gtest/gtest.h>

#include "DynamicArray.h"

// Test initialization

TEST(DynamicArrayTest, Initialization) {

DynamicArray<int> arr;

EXPECT\_EQ(arr.size(), 0);

EXPECT\_EQ(arr.capacity(), 1); // Assuming initial capacity is 1

EXPECT\_TRUE(arr.empty());

}

// Test push\_back and size

TEST(DynamicArrayTest, PushBackAndSize) {

DynamicArray<int> arr;

arr.push\_back(1);

EXPECT\_EQ(arr.size(), 1);

EXPECT\_EQ(arr[0], 1);

arr.push\_back(2);

EXPECT\_EQ(arr.size(), 2);

EXPECT\_EQ(arr[1], 2);

}

// Test resizing

TEST(DynamicArrayTest, Resizing) {

DynamicArray<int> arr;

for (int i = 0; i < 10; ++i) {

arr.push\_back(i);

}

EXPECT\_EQ(arr.size(), 10);

EXPECT\_GE(arr.capacity(), 10); // Capacity should be at least 10

for (int i = 0; i < 10; ++i) {

EXPECT\_EQ(arr[i], i);

}

}

// Test pop\_back

TEST(DynamicArrayTest, PopBack) {

DynamicArray<int> arr;

arr.push\_back(1);

arr.push\_back(2);

EXPECT\_EQ(arr.size(), 2);

arr.pop\_back();

EXPECT\_EQ(arr.size(), 1);

EXPECT\_EQ(arr[0], 1);

arr.pop\_back();

EXPECT\_EQ(arr.size(), 0);

EXPECT\_TRUE(arr.empty());

}

// Test operator[]

TEST(DynamicArrayTest, IndexOperator) {

DynamicArray<int> arr;

arr.push\_back(10);

arr.push\_back(20);

EXPECT\_EQ(arr[0], 10);

EXPECT\_EQ(arr[1], 20);

arr[0] = 30;

EXPECT\_EQ(arr[0], 30);

}

// Test clear

TEST(DynamicArrayTest, Clear) {

DynamicArray<int> arr;

arr.push\_back(1);

arr.push\_back(2);

arr.push\_back(3);

arr.clear();

EXPECT\_EQ(arr.size(), 0);

EXPECT\_TRUE(arr.empty());

}

// Test out of bounds access

TEST(DynamicArrayTest, OutOfBoundsAccess) {

DynamicArray<int> arr;

arr.push\_back(1);

EXPECT\_THROW(arr[1], std::out\_of\_range);

}

// Test with custom data type

struct Point {

int x, y;

bool operator==(const Point& other) const {

return x == other.x && y == other.y;

}

};

TEST(DynamicArrayTest, CustomDataType) {

DynamicArray<Point> arr;

arr.push\_back({1, 2});

arr.push\_back({3, 4});

EXPECT\_EQ(arr[0], Point{1, 2});

EXPECT\_EQ(arr[1], Point{3, 4});

}