МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**Национальный исследовательский ядерный университет «МИФИ»**

****

**Институт интеллектуальных кибернетических систем**

**КАФЕДРА «КИБЕРНЕТИКА» (№22)**

**Технологии промышленной разработки программного обеспечения Сертификация**

|  |
| --- |
| Лабораторная работа №3-4 |

|  |  |  |  |
| --- | --- | --- | --- |
| Группа | М23-524 |  | |
| Студент |  |  | Леонов В.В. |
|  | (подпись) | (ФИО) | |
| Преподаватель |  |  | Гагарин А.П. |
|  | (подпись) | (ФИО) | |
| Оценка преподавателя |  |  |  |
|  | (оценка) | (дата) | |

**Москва 2024**

**Спецификация модуля**

**Назначение.** Модуль оплаты (payment) предназначен для обработки финансовых операций, связанных с заказами в ресторане. Основная цель модуля — управление процессами расчета общей суммы заказа, регистрации платежей, их обновления и возвратов. Модуль обеспечивает взаимодействие между системой хранения данных платежей и системой меню ресторана, предоставляя прозрачный и безопасный процесс оплаты.

**Функции.** Модуль предоставляет следующие ключевые функции:

1. **Расчет общей суммы заказа** (CalculateTotal):
   * Рассчитывает общую стоимость заказа на основе списка заказанных товаров и их количества.
   * Получает цены на позиции меню через интерфейс Menu.
   * Обрабатывает параллельное вычисление цен для повышения производительности.
2. **Обработка платежа** (ProcessPayment):
   * Рассчитывает итоговую стоимость заказа и инициирует создание записи о платеже.
   * Сохраняет информацию о платеже в хранилище через интерфейс Storage.
3. **Возврат платежа** (RefundPayment):
   * Выполняет возврат платежа для указанного заказа, проверяя его статус.
   * Обновляет статус платежа на "Refunded".

**Основные компоненты.**

1. **Интерфейсы:**
   * Menu — отвечает за получение цен на позиции меню.
   * Storage — отвечает за хранение и управление данными о платежах.
2. **Типы данных:**
   * Payment — структура, описывающая платеж (ID заказа, сумма, метод оплаты, статус).
   * OrderItem — структура, описывающая элемент заказа (ID позиции меню и количество).
3. **Константы** cодержат статусы платежей: Pending, Completed, Refunded.
4. **Сервис - о**сновной компонент, предоставляющий бизнес-логику для работы с платежами.

**Место в проектной модели.** Модуль оплаты является частью бэкенд-архитектуры ресторана и взаимодействует с другими модулями:

* **Меню**: Использует интерфейс Menu для получения информации о ценах на позиции меню.
* **Хранилище**: Работает с Storage для записи, получения, обновления и удаления данных о платежах.
* **API**: Предоставляет функциональность для внешнего взаимодействия (например, через контроллеры или обработчики запросов).

**Программный код модуля**

package payment

//go:generate mockgen -destination mocks/storage.go -package mocks . Storage

//go:generate mockgen -destination mocks/menu.go -package mocks . Menu

import (

    "context"

    "fmt"

    "sync"

    "github.com/go-faster/errors"

)

const (

    PaymentPending   = "Pending"

    PaymentCompleted = "Completed"

    PaymentRefunded  = "Refunded"

)

type Payment struct {

    OrderID int

    Amount  float64

    Method  string

    Status  string

}

type OrderItem struct {

    MenuItemID int

    Quantity   int

}

type Menu interface {

    GetItemPrice(ctx context.Context, itemID int) (float64, error)

}

type Storage interface {

    CreatePayment(ctx context.Context, orderID int, amount float64, method string) error

    GetPayment(ctx context.Context, orderID int) (Payment, error)

    UpdatePayment(ctx context.Context, orderID int, status string) error

    DeletePayment(ctx context.Context, orderID int) error

}

type Service struct {

    storage Storage

    menu    Menu

}

func New(storage Storage, menu Menu) \*Service {

    return &Service{

        storage: storage,

        menu:    menu,

    }

}

func (s \*Service) CalculateTotal(ctx context.Context, orderID int, items []OrderItem) (float64, error) {

    const op = "service.CalculateTotal"

    total := 0.0

    var wg sync.WaitGroup

    errChan := make(chan error, len(items))

    priceChan := make(chan float64, len(items))

    for \_, item := range items {

        wg.Add(1)

        go func(item OrderItem) {

            defer wg.Done()

            price, err := s.menu.GetItemPrice(ctx, item.MenuItemID)

            if err != nil {

                errChan <- fmt.Errorf("fetch price for item %d: %v", item.MenuItemID, err)

                return

            }

            priceChan <- float64(item.Quantity) \* price

        }(item)

    }

    wg.Wait()

    close(errChan)

    close(priceChan)

    for err := range errChan {

        return 0, errors.Wrap(err, op)

    }

    for price := range priceChan {

        total += price

    }

    return total, nil

}

func (s \*Service) ProcessPayment(ctx context.Context, orderID int, items []OrderItem, method string) (bool, error) {

    const op = "service.ProcessPayment"

    total, err := s.CalculateTotal(ctx, orderID, items)

    if err != nil {

        return false, errors.Wrap(err, op)

    }

    err = s.storage.CreatePayment(ctx, orderID, total, method)

    if err != nil {

        return false, errors.Wrap(err, op)

    }

    return true, nil

}

func (s \*Service) RefundPayment(ctx context.Context, orderID int) error {

    const op = "service.RefundPayment"

    payment, err := s.storage.GetPayment(ctx, orderID)

    if err != nil {

        return errors.Wrap(err, op)

    }

    if payment.Status != PaymentCompleted {

        return fmt.Errorf("%s: status is %s", op, payment.Status)

    }

    err = s.storage.UpdatePayment(ctx, orderID, PaymentRefunded)

    if err != nil {

        return errors.Wrap(err, op)

    }

    return nil

}

**Тестирование**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестируемая функция | Сценарий тестирования | Ожидаемый результат |
| 1 | CalculateTotal | Корректная работа по расчету полной стоимости заказа | Расчет стоимости заказа без ошибок |
| 2 | CalculateTotal | Произошла ошибка при получении стоимости позиции заказа | Возврат произошедшей ошибки |
| 3 | ProcessPayment | Корректная работа по созданию платежа | Отсутствие ошибок |
| 4 | ProcessPayment | Произошла ошибка при обращении в базу данных | Возврат произошедшей ошибки |
| 5 | ProcessPayment | Произошла ошибка при получении полной стоимости заказа | Возврат произошедшей ошибки |
| 6 | RefundPayment | Корректная работа по возврату платежа | Отсутствие ошибок |
| 7 | RefundPayment | Платеж не может быть отменен ввиду его статуса | Возврат произошедшей ошибки |
| 8 | RefundPayment | Произошла ошибка при обновлении базы данных | Возврат произошедшей ошибки |
| 9 | RefundPayment | Произошла ошибка при чтении из базы данных | Возврат произошедшей ошибки |

**Программный код тестов**

package payment\_test

import (

    "context"

    "testing"

    "github.com/go-faster/errors"

    "github.com/stretchr/testify/suite"

    "go.uber.org/mock/gomock"

    "restoraunt/internal/service/payment"

    "restoraunt/internal/service/payment/mocks"

)

type ServiceSuite struct {

    suite.Suite

    service \*payment.Service

    storage \*mocks.MockStorage

    menu    \*mocks.MockMenu

}

func (s \*ServiceSuite) SetupTest() {

    ctrl := gomock.NewController(s.T())

    s.storage = mocks.NewMockStorage(ctrl)

    s.menu = mocks.NewMockMenu(ctrl)

    s.service = payment.New(

        s.storage,

        s.menu,

    )

}

func (s \*ServiceSuite) TestCalculateTotal\_Success() {

    ctx := context.Background()

    items := []payment.OrderItem{

        {MenuItemID: 1, Quantity: 2},

        {MenuItemID: 2, Quantity: 3},

    }

    s.menu.EXPECT().GetItemPrice(ctx, 1).Return(10.0, nil)

    s.menu.EXPECT().GetItemPrice(ctx, 2).Return(20.0, nil)

    total, err := s.service.CalculateTotal(ctx, 1, items)

    s.NoError(err)

    s.Equal(80.0, total)

}

func (s \*ServiceSuite) TestCalculateTotal\_ErrorFetchingPrice() {

    ctx := context.Background()

    items := []payment.OrderItem{

        {MenuItemID: 1, Quantity: 2},

    }

    s.menu.EXPECT().GetItemPrice(ctx, 1).Return(0.0, context.DeadlineExceeded)

    total, err := s.service.CalculateTotal(ctx, 1, items)

    s.EqualError(err, "service.CalculateTotal: fetch price for item 1: context deadline exceeded")

    s.Zero(total)

}

func (s \*ServiceSuite) TestProcessPayment\_Success() {

    ctx := context.Background()

    items := []payment.OrderItem{

        {MenuItemID: 1, Quantity: 2},

    }

    s.menu.EXPECT().GetItemPrice(ctx, 1).Return(10.0, nil)

    s.storage.EXPECT().CreatePayment(ctx, 1, 20.0, "card").Return(nil)

    success, err := s.service.ProcessPayment(ctx, 1, items, "card")

    s.NoError(err)

    s.True(success)

}

func (s \*ServiceSuite) TestProcessPayment\_ErrorCreatingPayment() {

    ctx := context.Background()

    items := []payment.OrderItem{

        {MenuItemID: 1, Quantity: 2},

    }

    s.menu.EXPECT().GetItemPrice(ctx, 1).Return(10.0, nil)

    s.storage.EXPECT().CreatePayment(ctx, 1, 20.0, "card").Return(context.DeadlineExceeded)

    success, err := s.service.ProcessPayment(ctx, 1, items, "card")

    s.EqualError(err, "service.ProcessPayment: context deadline exceeded")

    s.Zero(success)

}

func (s \*ServiceSuite) TestProcessPayment\_ErrorCalculateTotal() {

    ctx := context.Background()

    items := []payment.OrderItem{

        {MenuItemID: 1, Quantity: 2},

    }

    s.menu.EXPECT().GetItemPrice(ctx, 1).Return(10.0, errors.New("some error"))

    success, err := s.service.ProcessPayment(ctx, 1, items, "card")

    s.EqualError(err, "service.ProcessPayment: service.CalculateTotal: fetch price for item 1: some error")

    s.False(success)

}

func (s \*ServiceSuite) TestRefundPayment\_Success() {

    ctx := context.Background()

    paymentRecord := payment.Payment{

        OrderID: 1,

        Amount:  20.0,

        Method:  "card",

        Status:  payment.PaymentCompleted,

    }

    s.storage.EXPECT().GetPayment(ctx, 1).Return(paymentRecord, nil)

    s.storage.EXPECT().UpdatePayment(ctx, 1, payment.PaymentRefunded).Return(nil)

    err := s.service.RefundPayment(ctx, 1)

    s.NoError(err)

}

func (s \*ServiceSuite) TestRefundPayment\_ErrorInvalidStatus() {

    ctx := context.Background()

    paymentRecord := payment.Payment{

        OrderID: 1,

        Amount:  20.0,

        Method:  "card",

        Status:  payment.PaymentPending,

    }

    s.storage.EXPECT().GetPayment(ctx, 1).Return(paymentRecord, nil)

    err := s.service.RefundPayment(ctx, 1)

    s.EqualError(err, "service.RefundPayment: status is Pending")

}

func (s \*ServiceSuite) TestRefundPayment\_ErrorUpdatePayment() {

    ctx := context.Background()

    paymentRecord := payment.Payment{

        OrderID: 1,

        Amount:  20.0,

        Method:  "card",

        Status:  payment.PaymentCompleted,

    }

    s.storage.EXPECT().GetPayment(ctx, 1).Return(paymentRecord, nil)

    s.storage.EXPECT().UpdatePayment(ctx, 1, payment.PaymentRefunded).Return(errors.New("some error"))

    err := s.service.RefundPayment(ctx, 1)

    s.EqualError(err, "service.RefundPayment: some error")

}

func (s \*ServiceSuite) TestRefundPayment\_ErrorGetPayment() {

    ctx := context.Background()

    s.storage.EXPECT().GetPayment(ctx, 1).Return(payment.Payment{}, errors.New("some error"))

    err := s.service.RefundPayment(ctx, 1)

    s.EqualError(err, "service.RefundPayment: some error")

}

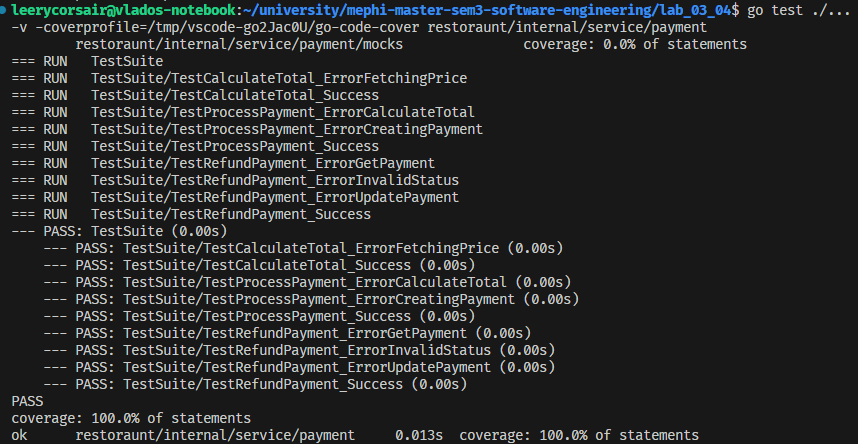
func TestSuite(t \*testing.T) {

    suite.Run(t, new(ServiceSuite))

}

**Проведение тестирования**

Тесты выполняются в автоматизированном режиме с помощью встроенного в стандартную версию Go инструмента тестирования.



**Заключение**

Тестирование показало, что система корректно выполняет все заявленные функции. Все тест-кейсы пройдены успешно, ошибок и критических сбоев не выявлено.

**Спецификация требований программного продукта**

**Программный продукт:**  
«Модуль оплаты для приложения ресторана»

**Заявитель:**  
Леонов Владислав Вячеславович

**Эксперт:**  
Леонов Владислав Вячеславович

**Дата заключения:**  
10.12.2024

**Общие требования**

1.1. Программный продукт должен соответствовать требованиям действующих стандартов и нормативных документов, регулирующих разработку и эксплуатацию программного обеспечения:

* **ISO/IEC 25010:2011** — «Системы и программная инженерия — Модели качества систем и программного обеспечения».
* **ГОСТ 28195-99** — «Программные средства. Документы, используемые в процессе разработки».
* **ГОСТ Р 56939-2016** — «Программное обеспечение. Общие требования к качеству».

1.2. Программный продукт должен обеспечивать соответствие современным требованиям безопасности, включая защиту данных пользователей и минимизацию рисков взлома или потери данных.

**Функциональные требования**

2.1. Функциональность

2.1.1. Программный продукт должен выполнять все заявленные в техническом задании функции в полном объеме и с высокой точностью.  
2.1.2. Продукт должен корректно обрабатывать пользовательские запросы, в том числе те, которые выходят за рамки стандартных сценариев.  
2.1.3. Все функции должны быть реализованы через программные интерфейсы (API), поддерживая интеграцию с другими системами.  
2.1.4. Должна быть предусмотрена обработка транзакций с учётом возможных прерываний для обеспечения целостности данных.

2.2. Корректность работы

2.2.1. Программный продукт должен обеспечивать точные вычисления для всех операций, включая расчёт стоимости заказа, учёт скидок, налогов и возвратов.  
2.2.2. Все ошибки должны быть обработаны с предоставлением пользователю информативных сообщений, включая рекомендации по исправлению.  
2.2.3. Исключительные ситуации (например, потеря соединения с базой данных) должны обрабатываться корректно, без потери данных или нарушения целостности системы.

2.3. Интерактивное взаимодействие

2.3.1. Продукт должен обеспечивать удобное взаимодействие с пользователем, минимизируя количество шагов для выполнения каждой операции.  
2.3.2. Интерфейс взаимодействия должен быть интуитивно понятным, включая автоматическое заполнение повторяющихся полей и предоставление подсказок.  
2.3.3. Должна быть предусмотрена возможность интеграции с внешними платежными системами для автоматической обработки оплат.

**Нефункциональные требования**

3.1. Производительность

3.1.1. Время отклика системы на стандартные запросы (например, расчёт стоимости заказа) не должно превышать 1 секунды.  
3.1.2. Программный продукт должен обеспечивать обработку до 1000 запросов в минуту без ухудшения производительности.

3.2. Надежность и устойчивость к отказам

3.2.1. Программный продукт должен обеспечивать стабильную работу в условиях повышенной нагрузки и сохранять данные в случае непредвиденного завершения операций.  
3.2.2. Должна быть реализована функция автоматического восстановления данных после сбоя.

3.3. Масштабируемость

3.3.1. Архитектура продукта должна предусматривать возможность добавления новых функций и модулей без значительных изменений в существующем коде.  
3.3.2. Система должна поддерживать работу в распределённой среде для повышения производительности и надёжности.

**Документирование**

4.1. Документация на продукт должна содержать:

* Подробное описание архитектуры и функциональности продукта.
* Руководство по установке, настройке и эксплуатации.
* Описание всех возможных ошибок и способы их устранения.

4.2. Документирование тестирования:

* Все тестовые сценарии должны быть описаны с указанием входных данных, шагов выполнения, ожидаемых результатов и возможных отклонений.
* Описание тестов должно быть структурировано таким образом, чтобы они могли быть легко воспроизведены для проверки корректности работы системы.

4.3. Обновления документации должны сопровождать каждую новую версию программного продукта, с указанием изменений и их влияния на текущую функциональность.

**План тестирования программного продукта**

**Программный продукт:**  
«Модуль оплаты для приложения ресторана»

**Заявитель:**  
Леонов Владислав Вячеславович

**Эксперт:**  
Леонов Владислав Вячеславович

**Дата заключения:**  
10.12.2024

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестируемая функция | Сценарий тестирования | Ожидаемый результат |
| 1 | CalculateTotal | Корректная работа по расчету полной стоимости заказа | Расчет стоимости заказа без ошибок |
| 2 | CalculateTotal | Произошла ошибка при получении стоимости позиции заказа | Возврат произошедшей ошибки |
| 3 | ProcessPayment | Корректная работа по созданию платежа | Отсутствие ошибок |
| 4 | ProcessPayment | Произошла ошибка при обращении в базу данных | Возврат произошедшей ошибки |
| 5 | ProcessPayment | Произошла ошибка при получении полной стоимости заказа | Возврат произошедшей ошибки |
| 6 | RefundPayment | Корректная работа по возврату платежа | Отсутствие ошибок |
| 7 | RefundPayment | Платеж не может быть отменен ввиду его статуса | Возврат произошедшей ошибки |
| 8 | RefundPayment | Произошла ошибка при обновлении базы данных | Возврат произошедшей ошибки |
| 9 | RefundPayment | Произошла ошибка при чтении из базы данных | Возврат произошедшей ошибки |

**Отчет о тестировании программного продукта**

**Программный продукт:**  
«Модуль оплаты для приложения ресторана»

**Заявитель:**  
Леонов Владислав Вячеславович

**Эксперт:**  
Леонов Владислав Вячеславович

**Дата заключения:**  
10.12.2024

**Автоматизация тестовых сценариев**

Все ключевые сценарии тестирования охвачены автоматическими тестами, и выполнение всех тест-кейсов прошло успешно.

**Оценка адекватности среды для целей тестирования**

Среда тестирования признана соответствующей целям сертификации, поскольку она предоставляет необходимые условия для выполнения всех операций с платежами. Это включает расчёт общей стоимости заказа, создание записей о платежах, обновление их статусов и обработку возвратов.

**Оценка полноты плана тестирования по составу тестов**

План тестирования охватывает все основные сценарии, включая расчёт общей суммы заказа, обработку платежей с различными методами, возврат платежей и обработку ошибок (например, недоступность интерфейсов). Также были протестированы исключительные ситуации, такие как некорректные входные данные или запросы на возврат для неподходящего статуса платежа. Дополнительные тесты с контролируемыми данными позволяют проверять корректность работы модуля при различных наборах входных данных.

**Оценка полноты плана тестирования по длительности**

Тестирование охватывает все критически важные функциональные аспекты модуля оплаты. Каждое тестирование выполняется в пределах разумного времени, соответствующего сложности операций, таких как параллельный расчёт стоимости или обработка многократных вызовов к интерфейсам. Время выполнения тестов оптимизировано для обеспечения быстроты и эффективности.

**Оценка качества документирования тестов и их воспроизводимости**

Тестовые сценарии хорошо документированы, с чётким описанием шагов и ожидаемых результатов. Используемые тестовые данные являются стандартными и управляемыми, что обеспечивает лёгкую воспроизводимость тестов. Все тесты могут быть повторно выполнены в любых условиях, благодаря точной фиксации требований и использования известных условий среды.

**Оценка соответствия среды тестирования условиям эксплуатации**

Среда тестирования соответствует реальным условиям эксплуатации приложения ресторана. Она моделирует взаимодействие с интерфейсами меню и системы хранения данных, что позволяет симулировать обработку реальных заказов, создание платежей и выполнение возвратов.

**Оценка достаточности автоматизации при тестировании**

Модуль тестируется полностью автоматически с использованием современных фреймворков и инструментов. Это обеспечивает высокую скорость выполнения тестов, стабильность результатов и легкость их воспроизведения. Автоматизация охватывает все основные сценарии, включая расчёт стоимости, регистрацию платежей и возвраты, что позволяет исключить человеческий фактор и повысить надёжность системы.

**Заключение о сертификации программного продукта**

**Программный продукт:**  
«Модуль оплаты для приложения ресторана»

**Заявитель:**  
Леонов Владислав Вячеславович

**Эксперт:**  
Леонов Владислав Вячеславович

**Дата заключения:**  
10.12.2024

### 1. **Общие сведения**

Программный продукт «Модуль оплаты для приложения ресторана» был представлен на сертификацию с целью подтверждения его соответствия требованиям, установленным для программного обеспечения данной категории.

### 2. **Результаты сертификации**

На основании анализа предоставленных документов и результатов проведенных тестов, а также оценки среды тестирования, сделаны следующие выводы:

* **Функциональность**: Модуль корректно выполняет заявленные функции, включая расчет стоимости заказа, обработку платежей и возврат платежей.
* **Среда тестирования**:
  + Адекватна целям сертификации, все необходимые сценарии были проверены.
  + Симулирует реальную эксплуатацию приложения ресторана, включая интеграцию с меню и системой хранения данных.
* **План тестирования**:
  + Достаточно полный, охватывает как стандартные сценарии, так и исключительные случаи (например, ошибки получения цен, некорректные входные данные).
  + Тестовые кейсы документированы и могут быть воспроизведены.
* **Автоматизация тестирования**: Реализованная автоматизация позволяет эффективно и быстро проверять модуль.
* **Безопасность**: Модуль соответствует требованиям безопасности, предотвращает некорректные изменения данных и гарантирует целостность информации о платежах.

### 3. **Заключение**

Программный продукт «Модуль оплаты для приложения ресторана» прошел сертификацию и признан соответствующим всем предъявленным требованиям и стандартам, установленным для программного обеспечения данной категории.

**Рекомендации**:  
Программный продукт может быть рекомендован для внедрения в приложение ресторана с целью управления оплатами, а также для дальнейшей эксплуатации в условиях реальной среды.

### 4. **Заключительное решение**

На основании проведенной сертификации и анализа предоставленных документов, программный продукт «Модуль оплаты для приложения ресторана» признан готовым к эксплуатации и использованию в составе системы приложения ресторана.