Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»
Факультет программной инженерии и компьютерной техники

Вариант: Пасека
Курсовая работа этап №3
по дисциплине
Информационные системы

Выполнил студент группы P3312
Соколов Анатолий Владимирович
Пархоменко Кирилл Александрович
Преподаватель:
Бострикова Дарья Константиновна

г. Санкт-Петербург
2024г.

Содержание

1 Отчет третьей части **2**
1.1 Изобразить диаграмму классов, представляющую общую архитектуру системы. 2
1.1.1 Реализовать уровень хранения информационной системы на основе разработанной на предыдущем этапе базы данных. 3
1.1.2 Конфигурация postgresql оператора 3
1.1.3 Значения для helm оператора 3
1.2 При реализации уровня хранения должны использоваться функции/процедуры, созданные на втором этапе с помощью pl/pgsql. Нельзя замещать их использование альтернативной реализацией аналогичных запросов на уровне хранения информационной системы. 4
1.3 Использование функций/процедур 4
1.4 Реализация уровеня бизнес-логики 7
1.5 Пример авторизации 9
1.6 Вызов функций внутри psql 10

1 Отчет третьей части

1.1 Изобразить диаграмму классов, представляющую общую архитектуру системы.

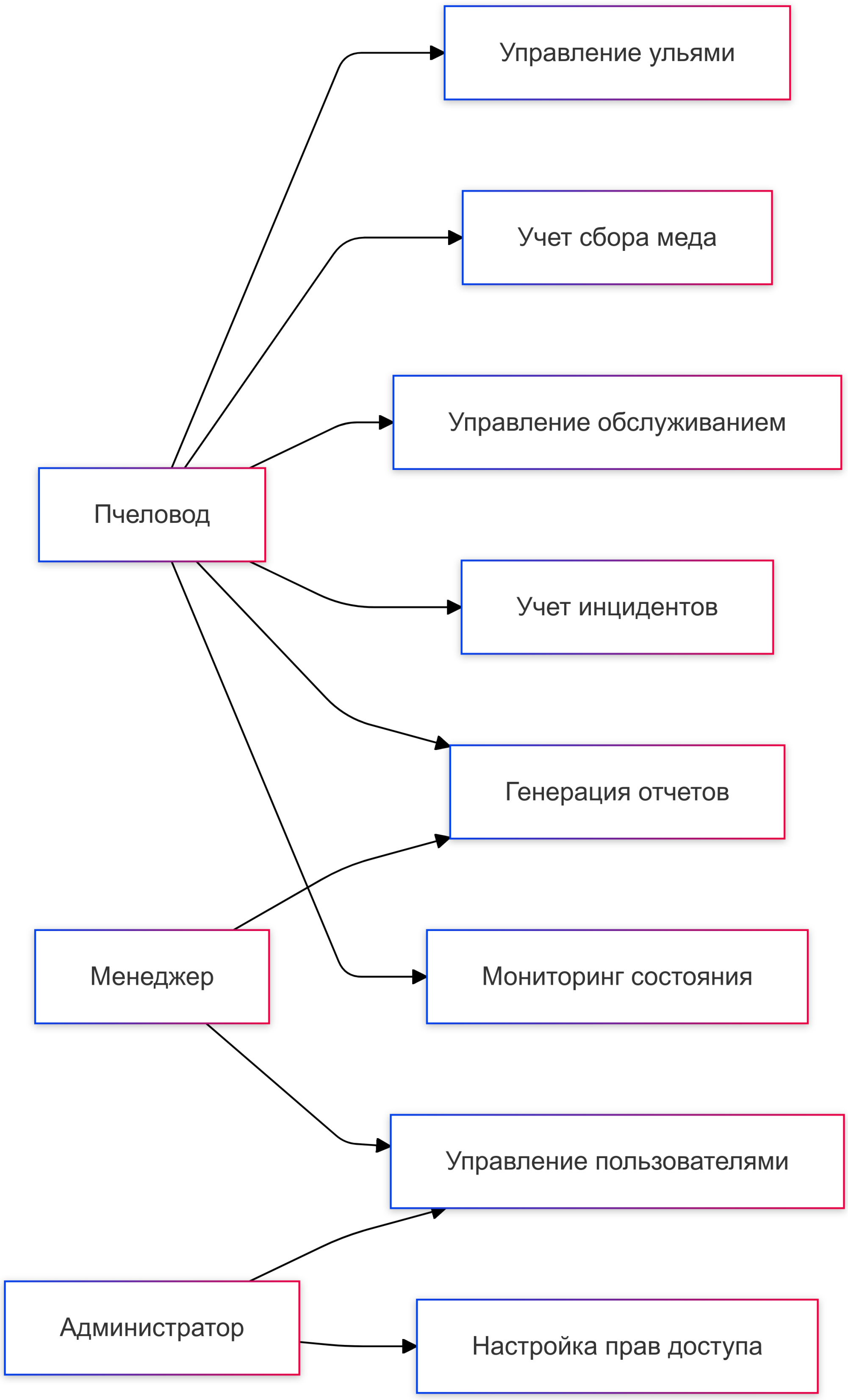

Рис. 1: ER диаграмма

На языке go нет классов.

1.1.1 Реализовать уровень хранения информационной системы на основе разработанной на предыдущем этапе базы данных.

1.1.2 Конфигурация postgresql оператора

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgresql
  namespace: { { .Values.cloudnativepg.namespace } }
spec:
  instances: { { .Values.cloudnativepg.instances } }
  imageName: { { .Values.cloudnativepg.imageName } }
  imagePullPolicy: { { .Values.cloudnativepg.imagePullPolicy } }
  primaryUpdateStrategy: unsupervised
  storage:
    size: { { .Values.cloudnativepg.storage.size } }
    storageClass: { { .Values.cloudnativepg.storage.storageClass } }
  superuserSecret:
    name: { { .Values.cloudnativepg.superuserSecret.name } }
  bootstrap:
    initdb:
      database: { { .Values.cloudnativepg.bootstrap.initdb.database } }
      owner: { { .Values.cloudnativepg.bootstrap.initdb.owner } }
  postgresql:
    parameters:
      max_connections: ”1000”
      shared_buffers: 256MB
  resources:
    requests:
      cpu: { { .Values.cloudnativepg.resources.requests.cpu } }
      memory: { { .Values.cloudnativepg.resources.requests.memory } }
    limits:
      cpu: { { .Values.cloudnativepg.resources.limits.cpu } }
      memory: { { .Values.cloudnativepg.resources.limits.memory } }
```

1.1.3 Значения для helm оператора

```yaml
namespace: beesbiz-data
clusterScoped: false

cloudnativepg:
  namespace: beesbiz-data
  instances: 3
  imageName: ghcr.io/cloudnative-pg/postgresql:14.7
  imagePullPolicy: IfNotPresent
  resources:
    requests:
      cpu: ”500m”
      memory: ”1Gi”
    limits:
      cpu: ”2”
      memory: ”2Gi”
  storage:
    size: 2Gi
    storageClass: ”standard”
  superuserSecret:
    name: postgresql-superuser
    namespace: beesbiz-data
  bootstrap:
    initdb:
      database: postgres
      owner: postgres
```

1.2 При реализации уровня хранения должны использоваться функции/процедуры, созданные на втором этапе с помощью pl/pgsql. Нельзя замещать их использование альтернативной реализацией аналогичных запросов на уровне хранения информационной системы.

```go

func (db *DB) InitSchema(pathToScripts string, sqlFiles []string) error {
	for _, file := range sqlFiles {
		filePath := filepath.Join(pathToScripts, file)
		zap.L().Info("Loading SQL file", zap.String("file", file))
		if err := db.executeSQLFile(filePath); err != nil {
			zap.L().Error("Failed to execute SQL file", zap.String("file", file), zap.Error(err))
			return fmt.Errorf("error executing SQL file %s: %w", file, err)
		}
		zap.L().Info("Successfully executed SQL file", zap.String("file", file))
	}

	zap.L().Info("All SQL files executed successfully")
	return nil
}

func (db *DB) executeSQLFile(filePath string) error {
	content, err := os.ReadFile(filePath)
	if err != nil {
		return fmt.Errorf("error reading SQL file: %w", err)
	}

	_, err = db.Exec(string(content))
	if err != nil {
		return fmt.Errorf("error executing SQL: %w", err)
	}
	return nil
}

func (db *DB) ExecuteSQL(sql string) error {
	_, err := db.Exec(sql)
	if err != nil {
		return fmt.Errorf("error executing SQL: %w", err)
	}
	return nil
}
```

1.3 Использование функций/процедур

```proto
syntax = "proto3";

package bee_management;

import "google/protobuf/empty.proto";
```

```

option go_package = "github.com/orientallines/beesbiz/bee_management";

// Service Definition
service BeeManagementService {
  // 1. Get Total Honey Harvested
  rpc GetTotalHoneyHarvested(GetTotalHoneyHarvestedRequest)
     returns (GetTotalHoneyHarvestedResponse) {}

  // 2. Add Observation
  rpc AddObservation(AddObservationRequest) returns (google.protobuf.Empty) {}

  // 3. Get Community Health Status
  rpc GetCommunityHealthStatus(GetCommunityHealthStatusRequest)
     returns (GetCommunityHealthStatusResponse) {}

  // 4. Update Hive Status
  rpc UpdateHiveStatus(UpdateHiveStatusRequest)
     returns (google.protobuf.Empty) {}

  // 5. Get Average Temperature
  rpc GetAvgTemperature(GetAvgTemperatureRequest)
     returns (GetAvgTemperatureResponse) {}

  // 6. Assign Maintenance Plan
  rpc AssignMaintenancePlan(AssignMaintenancePlanRequest)
     returns (google.protobuf.Empty) {}

  // 7. Check Region Access
  rpc HasRegionAccess(HasRegionAccessRequest)
     returns (HasRegionAccessResponse) {}

  // 8. Register Incident
  rpc RegisterIncident(RegisterIncidentRequest)
     returns (google.protobuf.Empty) {}

  // 9. Get Latest Sensor Reading
  rpc GetLatestSensorReading(GetLatestSensorReadingRequest)
     returns (GetLatestSensorReadingResponse) {}

  // 10. Create Production Report
  rpc CreateProductionReport(CreateProductionReportRequest)
     returns (google.protobuf.Empty) {}

  // 11. Set Region Access
  rpc SetRegionAccess(SetRegionAccessRequest) returns (google.protobuf.Empty) {}
}

// Message Definitions

// 1. GetTotalHoneyHarvested
message GetTotalHoneyHarvestedRequest {
  int32 hive_id = 1;
  string start_date = 2; // Format: YYYY-MM-DD
  string end_date = 3; // Format: YYYY-MM-DD
}

message GetTotalHoneyHarvestedResponse { double total_honey = 1; }

// 2. AddObservation
message AddObservationRequest {
  int32 hive_id = 1;
  string observation_date = 2; // Format: YYYY-MM-DD
  string description = 3;
  string recommendations = 4;
}

// 3. GetCommunityHealthStatus
message GetCommunityHealthStatusRequest { int32 community_id = 1; }

message GetCommunityHealthStatusResponse { string health_status = 1; }

// 4. UpdateHiveStatus
message UpdateHiveStatusRequest {
  int32 hive_id = 1;
  string new_status = 2;
}

// 5. GetAvgTemperature
message GetAvgTemperatureRequest {
  int32 region_id = 1;
  int32 days = 2;
}

message GetAvgTemperatureResponse { double avg_temperature = 1; }

// 6. AssignMaintenancePlan
message AssignMaintenancePlanRequest {
  int32 plan_id = 1;
  int32 user_id = 2;
}

// 7. HasRegionAccess
message HasRegionAccessRequest {
  int32 user_id = 1;
  int32 region_id = 2;
}

message HasRegionAccessResponse { bool has_access = 1; }

// 8. RegisterIncident
message RegisterIncidentRequest {
  int32 hive_id = 1;
  string incident_date = 2; // Format: YYYY-MM-DD
  string description = 3;
  string severity = 4;
}

// 9. GetLatestSensorReading
message GetLatestSensorReadingRequest {
  int32 hive_id = 1;
  string sensor_type = 2;
}

message GetLatestSensorReadingResponse {
  bytes value = 1;
  string timestamp = 2; // ISO 8601 format
}

// 10. CreateProductionReport
message CreateProductionReportRequest {
  int32 apiary_id = 1;
  string start_date = 2; // Format: YYYY-MM-DD
  string end_date = 3; // Format: YYYY-MM-DD
}

// 11. SetRegionAccess
message SetRegionAccessRequest {
  int32 user_id = 1;
  int32 region_id = 2;
}
```

1.4 Реализация уровеня бизнес-логики

```
   type Server struct {
      app *fiber.App
      db *database.DB
      jwtKey []byte
}

// NewServer creates a new Server
func NewServer(db *database.DB) *Server {
      return &Server{
            app: fiber.New(),
            db: db,
            jwtKey: []byte(config.GlobalConfig.JwtSecret),
      }
}

// SetupRoutes sets up the routes for the server
func (s *Server) SetupRoutes() {
      s.app.Use(requestid.New())
      // s.app.Use(logger.New(logger.Config{
      //  Format: "[${time}] ${status} - ${method} ${path}\n",
      // }))
      s.app.Use(healthcheck.New(healthcheck.Config{
            LivenessProbe: func(c *fiber.Ctx) bool {
                  return true
            },
            LivenessEndpoint: "/livez",
            ReadinessProbe: func(c *fiber.Ctx) bool {
                  return true
            },
            ReadinessEndpoint: "/readyz",
      }))

      auth := s.app.Group("/auth")

      auth.Post("/login", handlers.Login(s.db, s.jwtKey))
      auth.Post("/register", handlers.Register(s.db))

      api := s.app.Group("/api", jwtMiddleware(s.jwtKey))

      // Apiary routes
      apiary := api.Group("/apiary", roleMiddleware(types.Worker, types.Manager,
         ↪  types.Admin))

      apiary.Get("/:id", handlers.GetApiary(s.db))
      apiary.Post("/", handlers.CreateApiary(s.db))
      apiary.Put("/", handlers.UpdateApiary(s.db))
      apiary.Delete("/:id", handlers.DeleteApiary(s.db))
      apiary.Get("/", handlers.GetAllApiaries(s.db))
```

```go

	// Hive routes
	hive := api.Group(”/hive”, roleMiddleware(types.Worker, types.Manager, types.Admin))

	hive.Get(”/”, handlers.GetAllHives(s.db))
	hive.Post(”/”, handlers.CreateHive(s.db))
	hive.Put(”/”, handlers.UpdateHive(s.db))
	hive.Delete(”/:id”, handlers.DeleteHive(s.db))
	hive.Get(”/:apiaryID/hives”, handlers.GetAllHivesByApiaryID(s.db))

	// BeeCommunity routes
	beeCommunity := api.Group(”/bee-community”, roleMiddleware(types.Worker, types.Manager, types.Admin))

	beeCommunity.Get(”/”, handlers.GetAllBeeCommunities(s.db))
	beeCommunity.Post(”/”, handlers.CreateBeeCommunity(s.db))
	beeCommunity.Put(”/”, handlers.UpdateBeeCommunity(s.db))
	beeCommunity.Delete(”/:id”, handlers.DeleteBeeCommunity(s.db))
	beeCommunity.Get(”/:hiveID/bee-communities”, handlers.GetAllBeeCommunitiesByHiveID(s.db))

	// HoneyHarvest routes
	honeyHarvest := api.Group(”/honey-harvest”, roleMiddleware(types.Worker, types.Manager, types.Admin))

	honeyHarvest.Get(”/:id”, handlers.GetHoneyHarvest(s.db))
	honeyHarvest.Post(”/”, handlers.CreateHoneyHarvest(s.db))
	honeyHarvest.Put(”/”, handlers.UpdateHoneyHarvest(s.db))
	honeyHarvest.Delete(”/:id”, handlers.DeleteHoneyHarvest(s.db))
	honeyHarvest.Get(”/”, handlers.GetAllHoneyHarvests(s.db))

	// Region routes
	region := api.Group(”/region”, roleMiddleware(types.Manager, types.Admin))

	region.Get(”/:id”, handlers.GetRegion(s.db))
	region.Post(”/”, handlers.CreateRegion(s.db))
	region.Put(”/”, handlers.UpdateRegion(s.db))
	region.Delete(”/:id”, handlers.DeleteRegion(s.db))
	region.Get(”/”, handlers.GetAllRegions(s.db))

	// AllowedRegion routes
	allowedRegion := api.Group(”/allowed-region”, roleMiddleware(types.Manager, types.Admin))

	allowedRegion.Get(”/user/:id”, handlers.GetAllowedRegionsForUser(s.db))
	allowedRegion.Post(”/”, handlers.CreateAllowedRegion(s.db))
	allowedRegion.Put(”/”, handlers.UpdateAllowedRegion(s.db))
	allowedRegion.Delete(”/:id”, handlers.DeleteAllowedRegion(s.db))
	allowedRegion.Get(”/”, handlers.GetAllAllowedRegions(s.db))

	// RegionApiary routes
	regionApiary := api.Group(”/region-apiary”, roleMiddleware(types.Manager, types.Admin))

	regionApiary.Get(”/:id”, handlers.GetRegionApiary(s.db))
	regionApiary.Post(”/”, handlers.CreateRegionApiary(s.db))
	regionApiary.Put(”/”, handlers.UpdateRegionApiary(s.db))
	regionApiary.Delete(”/:id”, handlers.DeleteRegionApiary(s.db))
	regionApiary.Get(”/”, handlers.GetAllRegionApiaries(s.db))

	// User routes
	user := api.Group("/user", roleMiddleware(types.Admin, types.Manager))

	user.Get("/:id", handlers.GetUser(s.db))
	user.Post("/", handlers.CreateUser(s.db))
	user.Put("/", handlers.UpdateUser(s.db))
	user.Delete("/:id", handlers.DeleteUser(s.db))
	user.Get("/", handlers.GetAllUsers(s.db))

	// ProductionReport routes
	productionReport := api.Group("/production-report", roleMiddleware(types.
	    ↪ Manager, types.Worker, types.Admin))

	productionReport.Get("/:id", handlers.GetProductionReport(s.db))
	productionReport.Post("/", handlers.CreateProductionReport(s.db))
	productionReport.Put("/", handlers.UpdateProductionReport(s.db))
	productionReport.Delete("/:id", handlers.DeleteProductionReport(s.db))
	productionReport.Get("/", handlers.GetAllProductionReports(s.db))

	// Sensor routes
	sensor := api.Group("/sensor", roleMiddleware(types.Admin, types.Manager,
	    ↪ types.Worker))

	sensor.Get("/:id", handlers.GetSensor(s.db))
	sensor.Post("/", handlers.CreateSensor(s.db))
	sensor.Put("/", handlers.UpdateSensor(s.db))
	sensor.Delete("/:id", handlers.DeleteSensor(s.db))
	sensor.Get("/", handlers.GetAllSensors(s.db))

	// SensorReading routes
	sensorReading := api.Group("/sensor-reading", roleMiddleware(types.Admin,
	    ↪ types.Manager, types.Worker))

	sensorReading.Get("/:id", handlers.GetSensorReading(s.db))
	sensorReading.Post("/", handlers.CreateSensorReading(s.db))
	sensorReading.Put("/", handlers.UpdateSensorReading(s.db))
	sensorReading.Delete("/:id", handlers.DeleteSensorReading(s.db))
	sensorReading.Get("/", handlers.GetAllSensorReadings(s.db))

	// WeatherData routes
	weatherData := api.Group("/weather-data", roleMiddleware(types.Admin,
	    ↪ types.Manager, types.Worker))

	weatherData.Get("/:id", handlers.GetWeatherData(s.db))
	weatherData.Post("/", handlers.CreateWeatherData(s.db))
	weatherData.Put("/", handlers.UpdateWeatherData(s.db))
	weatherData.Delete("/:id", handlers.DeleteWeatherData(s.db))
	weatherData.Get("/", handlers.GetAllWeatherData(s.db))

}
```

1.5 Пример авторизации

```bash
BASE_URL="http://localhost:4040"
API_URL="${BASE_URL}/api"

curl -X POST "${BASE_URL}/auth/login" -H "Content-Type: application/json" -d '{"
    ↪ email_or_username": "john@example.com", "password": "password"}'

# Примерответа
# {"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MzAwMjgzOTQsInJvbGUiOiJXT1JLRVIiLCJ1c2VyX2lkIjoxMDF9.8rim7ZBdT-o8K1PpPqpg5obK3is1U30nSa2dB52bqRM"}%
curl -X POST "${API_URL}/apiary" \
  -H "Content-Type: application/json" \
  -d '{"location": "Test Location", "manager_id": 1, "establishment_date": "2023-01-01T15:04:05Z"}' \·
 -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE3MzAwMjEzNjYsInJvbGUiOiJXT1JLRVIiLCJ1c2VyX2lkIjoxMDF9.JP_KvMOAyivc2rJQnhC_ajgrwy9cJPjfdynLC6KbNSk"
# {"apiary_id":136,"location":"Test Location","manager_id":1,"establishment_date":"2023-01-01T00:00:00Z"}
```

1.6 Вызов функций внутри psql

```
  import * as grpc from "@grpc/grpc-js";
import * as protoLoader from "@grpc/proto-loader";
import path from "node:path";

// Define the path to the proto file
const PROTO_PATH = path.join(__dirname, "../../proto/bee_management.proto");

// Load the protobuf
const packageDefinition = protoLoader.loadSync(PROTO_PATH, {
  keepCase: true,
  longs: String,
  enums: String,
  defaults: true,
  oneofs: true,
});

// Load the package definition
const protoDescriptor = grpc.loadPackageDefinition(packageDefinition) as any;

// Get the BeeManagementService
const beeManagement = protoDescriptor.bee_management.BeeManagementService;

// Create a client instance
const client = new beeManagement("localhost:50051", grpc.credentials.createInsecure());

// Helper function to promisify client methods
function promisifyClientMethod(method: Function) {
  return (...args: any[]) => {
    return new Promise((resolve, reject) => {
      method(...args, (error: any, response: any) => {
        if (error) {
          reject(error);
        } else {
          resolve(response);
        }
      });
    });
  };
}

// Promisified client methods
const getTotalHoneyHarvested = promisifyClientMethod(client.GetTotalHoneyHarvested.bind(client));
```

```javascript
const addObservation = promisifyClientMethod(client.AddObservation.bind(client));
const getCommunityHealthStatus = promisifyClientMethod(
  client.GetCommunityHealthStatus.bind(client),
);
const updateHiveStatus = promisifyClientMethod(client.UpdateHiveStatus.bind(client));
const getAvgTemperature = promisifyClientMethod(client.GetAvgTemperature.bind(client));
const assignMaintenancePlan = promisifyClientMethod(client.AssignMaintenancePlan.bind(client));
const hasRegionAccess = promisifyClientMethod(client.HasRegionAccess.bind(client));
const registerIncident = promisifyClientMethod(client.RegisterIncident.bind(client));
const getLatestSensorReading = promisifyClientMethod(client.GetLatestSensorReading.bind(client));
const createProductionReport = promisifyClientMethod(client.CreateProductionReport.bind(client));

async function main() {
  try {
    // 1. Get Total Honey Harvested
    const totalHoney = await getTotalHoneyHarvested({
      hive_id: 1,
      start_date: "2023-01-01",
      end_date: "2024-12-31",
    });
    console.log("Total Honey Harvested:", totalHoney.total_honey);

    // 2. Add Observation
    const addObsResponse = await addObservation({
      hive_id: 1,
      observation_date: "2023-04-15",
      description: "Queen is healthy.",
      recommendations: "Continue current beekeeping practices.",
    });
    console.log("Add Observation Response:", addObsResponse);

    // 3. Get Community Health Status
    const communityHealth = await getCommunityHealthStatus({
      community_id: 1,
    });
    console.log("Community Health Status:", communityHealth.health_status);

    // 4. Update Hive Status
    const updateStatusResponse = await updateHiveStatus({
      hive_id: 1,
      new_status: "Active",
    });
    console.log("Update Hive Status Response:", updateStatusResponse);

    // 5. Get Average Temperature
    const avgTemp = await getAvgTemperature({
      region_id: 5,
      days: 30,
    });
    console.log("Average Temperature:", avgTemp.avg_temperature);

    // 6. Assign Maintenance Plan
    const assignPlanResponse = await assignMaintenancePlan({
```

```
    plan_id: 7,
    user_id: 3,
  });
  console.log("Assign Maintenance Plan Response:", assignPlanResponse);

  // 7. Has Region Access
  const regionAccess = await hasRegionAccess({
    user_id: 42,
    region_id: 5,
  });
  console.log("Has Region Access:", regionAccess.has_access);

  // 8. Register Incident
  const registerIncidentResponse = await registerIncident({
    hive_id: 1,
    incident_date: "2023-05-20",
    description: "Varroa mite infestation detected.",
    severity: "High",
  });
  console.log("Register Incident Response:", registerIncidentResponse);

  // 9. Get Latest Sensor Reading
  const latestSensor = await getLatestSensorReading({
    hive_id: 1,
    sensor_type: "humidity",
  });
  console.log("Latest Sensor Reading:", latestSensor);

  // 10. Create Production Report
  const createReportResponse = await createProductionReport({
    apiary_id: 1,
    start_date: "2023-01-01",
    end_date: "2023-06-30",
  });
  console.log("Create Production Report Response:", createReportResponse);
 } catch (error) {
  console.error("An error occurred:", error);
 } finally {
  client.close();
 }
}

main();
```