Etufillari – Leasing Backend Technical Assignment

This assignment simulates part of Etufillari's backend architecture. Your goal is to implement a small Leasing API module following the same folder structure, code quality, and architectural conventions used in our production system.

Context

Etufillari provides leasing (osamaksu / hire-purchase) solutions for company equipment. The goal of this task is to implement endpoints to manage leasing agreements and related payments. You'll implement this as a new module within our Azure Functions v3 TypeScript backend. Timebox your effort to about 2–3 hours.

Assignment Requirements

- Azure Functions v3 (Node.js 14).
- TypeScript strict mode enabled.
- Follow the given backend folder structure and conventions.
- Focus on clean architecture, precise typing, and good separation of concerns.
- Provide clear README and runnable local setup via Azure Functions Core Tools.

Functional Requirements

- POST /api/leases – Create a new leasing contract for a company.
- GET /api/leases/{id} – Fetch lease details, including remaining balance and payment schedule.
- POST /api/payments – Register a payment for an existing lease, updating remaining balance.
- GET /api/quote – Calculate a leasing quote (monthly payment, interest, fees) without persisting.

Expected Folder Structure

```
functions/
■■■ leases/
■   ■■■ function.json
■   ■■■ index.ts
■■■ payments/
■   ■■■ function.json
■   ■■■ index.ts
■■■ quote/
■   ■■■ function.json
■   ■■■ index.ts
■■■ health-check/
■   ■■■ function.json
■   ■■■ index.ts
src/
■■■ application/
■   ■■■ lease-service.ts
■   ■■■ payment-service.ts
■   ■■■ quote-service.ts
■■■ domain/
■   ■■■ lease.ts
■   ■■■ payment.ts
■   ■■■ quote.ts
■■■ handlers/
■   ■■■ lease/
■   ■   ■■■ create-lease.ts
■   ■   ■■■ get-lease.ts
■   ■■■ payment/
■   ■   ■■■ record-payment.ts
■   ■■■ quote/
■       ■■■ calculate-quote.ts
■■■ lib/
■   ■■■ prisma.ts
■   ■■■ validation.ts
■   ■■■ api-key-middleware.ts
■   ■■■ logger.ts
■■■ persistence/
■   ■■■ lease-repository.ts
■   ■■■ payment-repository.ts
■   ■■■ quote-repository.ts
■■■ index.ts
tests/
■■■ unit/
■   ■■■ domain/
■   ■   ■■■ lease.test.ts
■   ■   ■■■ payment.test.ts
■   ■■■ application/
■   ■   ■■■ lease-service.test.ts
■   ■   ■■■ payment-service.test.ts
■■■ integration/
    ■■■ lease-api.test.ts
    ■■■ payment-api.test.ts
```

Domain Models

```
type Money = number;

type LeaseInput = {
  companyId: string;
  itemId: string;
  price: Money;
  termMonths: number;
  nominalRatePct: number;
  startDate: string; // ISO date
  upfrontFee: Money;
  monthlyFee: Money;
};

type Lease = LeaseInput & {
  id: string;
  createdAt: string;
  schedule: Installment[];
  totals: { totalPayments: Money; totalInterest: Money; totalFees: Money };
};

type Installment = {
  period: number;
  dueDate: string;
  payment: Money;
  interest: Money;
  principal: Money;
  fee: Money;
  balanceAfter: Money;
};

type Payment = {
  id: string;
  leaseId: string;
  paidAt: string;
  amount: Money;
};
```

Non-Functional Requirements

- TypeScript strict mode; no `any` or `unknown`.
- Use Zod or equivalent validation library for request validation.
- API key authentication via x-api-key header (configurable in local.settings.json).
- Prisma-based repository or mocked JSON persistence.
- Unit tests for annuity calculation, schedule generation, and payment allocation logic.
- Integration test for lease creation + payment recording.
- Comprehensive README explaining local setup, endpoints, and Azure deployment steps.

Azure Setup

- Function runtime: v3 (Node.js 14).
- Storage account for Azure Functions.
- App settings: API_KEY, database connection string if using Prisma.
- Deploy via `func azure functionapp publish `.

■ Note: You can complete this challenge entirely with free Azure resources. The Azure Functions Consumption plan, Storage Account, and Application Insights all include generous free tiers. Alternatively, you can run everything locally using Azurite and Azure Functions Core Tools — deployment is optional.

Sample Configuration Files

function.json (for HTTP trigger)

```
{
  "bindings": [
    { "authLevel": "anonymous", "type": "httpTrigger", "direction": "in", "name": "req", "meth
    { "type": "http", "direction": "out", "name": "res" }
  ]
}
```

host.json

```
{
  "version": "2.0",
  "logging": { "applicationInsights": { "samplingSettings": { "isEnabled": true } } },
  "extensionBundle": { "id": "Microsoft.Azure.Functions.ExtensionBundle", "version": "[2.*, 3.
}
```

local.settings.json

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "UseDevelopmentStorage=true",
    "FUNCTIONS_WORKER_RUNTIME": "node",
    "API_KEY": "local-dev-key",
    "DATABASE_URL": "file:./prisma/dev.db"
  }
}
```

Submission & Evaluation

Deliver a GitHub repository or zip containing the full code, including tests and README. Ensure `npm test` and `func start` work locally without manual setup.

- Code correctness and API behavior.
- TypeScript strictness and type precision.
- Architectural alignment with Etufillari backend structure.
- Validation and error handling quality.
- Documentation and developer experience.
- Test quality (logic correctness + clarity).