Project Setup:

- Use PHP (preferably PHP 8.1 or later).
- Use a framework of your choice (e.g., Laravel, Symfony).
- Use Docker container to run the project.

Application Functionality:

- Create an application that calculates a payment schedule based on the provided product information.
- The product information should include:
    - productType: Products are grouped by Type.
    - productName: Name of the product.
    - productPrice: Object containing amount and currency.
    - productSoldDate: Date when the product was sold.

API Endpoints:

- Define the RESTful API for Payment Schedule Calculation
- Describe versioning approach for future versions of API with Breaking Changes

Payment Schedule Calculation:

- The payment schedule should be returned as JSON which contains:
    - amount: Payment amount.
    - currency: Currency of the payment.
    - dueDate: Due date for the payment.
- Implement rules for calculating the payment schedule based on productType and productSoldDate. For example:
- If the product is sold in June, the payment schedule is calculated into 2 installments where the 1st installment is 30% paid immediately and the rest is paid 3 months from the end of the current month.
- Prepare at least 3 different rules plus 1 fallback rule to be used whenever there is no other suitable rule.
- Rules can be defined in different currencies, but the response has to be in the same currency as request
- Requests can come with time in different timezone
- Payment schedule rules has to be immutable

Architecture requirements

- Use CQRS pattern to separate functional parts of application

- Implement Strategy pattern for different payment rules
- Use proper Value Object for manipulation with Money

Data Storage:

- Use a database of your choice (e.g., MySQL, PostgreSQL, SQLite) to store product information and payment schedules.
- Ensure the database schema is well-designed and normalized.
- Use DB Transactions
- Implement Repository pattern

Authentication:

- Implement basic authentication (e.g., API key, token-based authentication).
- Use authorization for more different user roles
- Secure app in accordance to OWASP top 10

Performance, Logging, Monitoring

- Define expected response times
- Log requests in structured format
- Log application errors in more levels by severity
- Collect data for metrics needed to define SLOs
- Specify monitoring endpoints (health check, readiness)

Error Handling:

- Implement proper error handling and return appropriate HTTP status codes.

Documentation:

- Provide API documentation (using Swagger/OpenAPI).
- Include instructions on how to set up and run the application.

Testing:

- Write unit tests, integration tests and API contract tests for your application (e.g., using PHPUnit, Behat).

Evaluation Criteria:

- Code quality and organization.
- Adherence to RESTful principles.
- Completeness and correctness of the implemented features.
- Quality of documentation and tests.