Tài liệu Yêu cầu Sản phẩm (PRD) cho URL Shortener

19/06/2025

Mục lục

1 Tổng quan

Tài liệu này mô tả yêu cầu cho dự án URL Shortener, một ứng dụng web cho phép người dùng rút gọn các URL dài thành các liên kết ngắn gọn, dễ chia sẻ. Hệ thống được xây dựng với các công nghệ hiện đại như Laravel (backend), React với TypeScript (frontend), Docker để đóng container, Redis để lưu cache, JWT để xác thực, và CI/CD để tự động hóa triển khai.

1.1 Mục tiêu

- Cung cấp dịch vụ rút gọn URL nhanh, đáng tin cậy và thân thiện với người dùng.
- Đảm bảo khả năng mở rộng và hiệu suất cao nhờ sử dụng cache và chỉ mục (indexing).
- Triển khai xác thực an toàn với JWT và refresh token.
- Tự động hóa phát triển, kiểm thử và triển khai với CI/CD.
- Tài liệu hóa API rõ ràng bằng Swagger.
- Triển khai ứng dụng trong môi trường container hóa.

1.2 Phạm vi

- **Tính năng cốt lõi**: Rút gọn URL, chuyển hướng, xác thực người dùng, quản lý liên kết.
- **Yêu cầu phi chức năng**: Hiệu suất cao, bảo mật, khả năng mở rộng, dễ bảo trì.
- **Ngoài phạm vi**: Phân tích nâng cao (ví dụ: theo dõi lượt nhấp), hỗ trợ tên miền tùy chỉnh.

2 Kiến trúc hệ thống

Ứng dụng tuân theo kiến trúc client-server với các thành phần sau:

- **Frontend**: React (TSX) cho ứng dụng giao diện đơn (SPA).
- **Backend**: Laravel (PHP) để phát triển API RESTful.
- **Cơ sở dữ liệu**: MySQL để lưu trữ lâu dài, với index để tối ưu truy vấn.
- **Cache**: Redis để lưu trữ URL truy cập thường xuyên.
- **Containerization**: Docker để đảm bảo môi trường phát triển và triển khai nhất quán.
- **Tài liệu API**: Swagger để tài liệu hóa API rõ ràng và tương tác.
- **Xác thực**: Laravel Sanctum để sử dụng JWT và refresh token.
- **Mẫu thiết kế**: Repository-Service pattern để viết mã sạch, dễ bảo trì.
- **CI/CD**: GitHub Actions để tự động hóa kiểm thử và triển khai.

- **Đăng nhập**: AWS ECS (Elastic Container Service) với ALB (Application Load Balancer).

2.1 Ngăn xếp công nghệ

- **Frontend**: React 18, TypeScript, Tailwind CSS, Axios.
- **Backend**: Laravel 11, PHP 8.3.
- **CDatabase**: MySQL 8.0.
- **Cache**: Redis 7.0.
- **Containerization**: Docker, Docker Compose.
- **Tài liệu API**: Swagger (OpenAPI 3.0).
- **Xác thực**: Laravel Sanctum để tạo JWT và refresh token.
- **CI/CD**: GitHub Actions.
- **Đăng nhập**: AWS ECS, AWS RDS (MySQL), AWS ElastiCache (Redis), Route 53 (DNS).
- **Giám sát**: AWS CloudWatch để ghi nhận log và số liệu.

3 Yêu cầu chức năng

3.1 Vai trò người dùng

- **Khách**: Có thể rút gọn URL mà không cần xác thực và truy cập URL ngắn.
- **Người dùng đã xác thực**: Có thể rút gọn URL, xem danh sách liên kết đã tạo, và xóa chúng.
-
- **Quản trị viên**: Có thể quản lý tất cả URL và người dùng (phần mở rộng trong tương lai).

3.2 Tính năng

3.2.1 Rút gọn URL

- Người dùng (khách hoặc đã xác thực) có thể nhập URL dài và nhận URL ngắn.
- URL ngắn sử dụng chuỗi mã hóa base62 (ví dụ: `http://short.url.vn/abc123`).
- URL của người dùng xác thực được liên kết với tài khoản để quản lý.

3.2.2 Chuyển hướng URL

- Truy cập URL ngắn sẽ chuyển hướng đến URL gốc với mã trạng thái 301.
- Chuyển hướng được lưu cache trong Redis để giảm độ trễ.

3.2.3 Xác thực người dùng

- **Đăng ký**: Người dùng tạo tài khoản với email và mật khẩu.
- **Đăng nhập**: Người dùng đăng nhập để nhận JWT và refresh token.
- **Đăng xuất**: Vô hiệu hóa token truy cập hiện tại.
- **Refresh Token**: Sử dụng refresh token để lấy token truy cập mới.
- Mật khẩu được băm bằng bcrypt của Laravel.

3.2.4 Quản lý liên kết (Người dùng xác thực)

- Xem danh sách các URL do người dùng tạo.
- Xóa URL (xóa mềm để duy trì tính toàn vẹn dữ liệu).

3.2.5 Tài liệu API

- Tất cả endpoint được tài liệu hóa bằng Swagger, truy cập tại `/api/docs`.

3.3 API Endpoint

Phương thức	**Endpoint**	**Mô tả**	**Xác thực**
POST	`/api/shorten`	Tạo URL ngắn	Tùy chọn
GET	`/api/urls`	Liệt kê URL của người dùng	Bắt buộc
DELETE	`/api/urls/{id}`	Xóa một URL	Bắt buộc
POST	`/api/register`	Đăng ký người dùng mới	Không
POST	`/api/login`	Đăng nhập và nhận JWT	Không
POST	`/api/logout`	Đăng xuất và vô hiệu hóa token	Bắt buộc
POST	`/api/refresh`	Làm mới JWT bằng refresh token	Bắt buộc
GET	`/{short_code}`	Chuyển hướng đến URL gốc	Không

4 Yêu cầu phi chức năng

4.1 Hiệu suất

- Độ trễ chuyển hướng: < 100ms (sử dụng cache Redis).
- Thời gian phản hồi API: < 200ms cho các yêu cầu xác thực.
- Truy vấn cơ sở dữ liệu được tối ưu với index trên `short_code` và `user_id`.

4.2 Khả năng mở rộng

- Mở rộng ngang với AWS ECS cho các container backend.
- Cụm Redis để mở rộng cache.
- Bản sao đọc MySQL cho lưu lượng đọc cao (xem xét trong tương lai).

4.3 Bảo mật

- Token JWT được ký bằng khóa an toàn, thời hạn ngắn (15 phút cho token truy cập, 7 ngày cho refresh token).
- Giới hạn tốc độ trên `/api/shorten` (ví dụ: 100 yêu cầu/giờ cho khách).
- Bảo vệ CSRF cho các biểu mẫu frontend.
- Xác thực và vệ sinh đầu vào để ngăn chặn SQL injection và XSS.
- Bắt buộc HTTPS cho tất cả yêu cầu.

4.4 Khả năng bảo trì

- Mẫu Repository-Service để tách logic nghiệp vụ khỏi controller.
- Kiểm thử đơn vị và tích hợp với PHPUnit (backend) và Jest (frontend).
- Kiểm tra mã với PHP CS Fixer và ESLint.
- Tài liệu Swagger rõ ràng cho API.

5 Thiết kế cơ sở dữ liệu

5.1 Bảng

- **users**
  - `id`: bigint, khóa chính
  - `email`: varchar, duy nhất
  - `password`: varchar (băm mã)
  - `name`: varchar
  - `created_at`, `updated_at`: timestamps
- **urls**
  - `id`: bigint, khóa chính
  - `user_id`: bigint, khóa ngoại (nullable cho URL khách)
  - `original_url`: text
  - `short_code`: varchar(6), duy nhất, có index
  - `created_at`, `updated_at`, `deleted_at`: timestamps
- **tokens** (cho refresh token)
  - `id`: bigint, khóa chính
  - `user_id`: bigint, khóa ngoại
  - `token`: text, duy nhất
  - `expires_at`: datetime

- `created_at`: timestamp

5.2 Chỉ mục (Indexing)

- `urls.short_code`: Index duy nhất để tra cứu nhanh.
- `urls.user_id`: Index cho các truy vấn theo người dùng.

6 Mẫu thiết kế

- **Repository Pattern**: Trừu tượng hóa thao tác cơ sở dữ liệu (ví dụ: `UrlRepository` cho các thao tác CRUD trên URL).
- **Service Pattern**: Đóng gói logic nghiệp vụ (ví dụ: `UrlService` cho logic rút gọn, xác thực, và lưu cache).

Ví dụ:

```
class UrlService {
    protected $urlRepository;
    public function __construct(UrlRepository $urlRepository) {
        $this->urlRepository = $urlRepository;
    }
    public function shorten(string $originalUrl, ?User $user): Url {
        // Xác thực, tạo mã ngắn, lưu vào DB, lưu cache trong Redis
    }
}
```

7 Thiết lập phát triển

7.1 Môi trường cục bộ

- Sử dụng Docker Compose để chạy:
  - Laravel (PHP-FPM + Nginx)
  - MySQL
  - Redis
  - Máy chủ phát triển React
- Ví dụ `đây là` *compose.yml* :

```
version: '3.8'
services:
  app:
    build: ./docker/php
    volumes:
      - .:/var/www/html
    depends_on:
      - mysql
      - redis
```

```
  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./docker/nginx/conf.d:/etc/nginx/conf.d
  mysql:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - mysql-data:/var/lib/mysql
  redis:
    image: redis:7.0-alpine
  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
volumes:
  mysql-data:
```

7.2 Các bước cài đặt

1. Sao chép kho lưu trữ: `git clone <repo-url>`.
2. Sao chép `.env.example` thành `.env` và cấu hình cài đặt cơ sở dữ liệu/cache.
3. Chạy `docker-compose up -d`.
4. Cài đặt phụ thuộc backend: `docker exec app composer install`.
5. Cài đặt phụ thuộc frontend: `docker exec frontend npm install`.
6. Chạy migration: `docker exec app php artisan migrate`.
7. Khởi động frontend: `docker exec frontend npm start`.

8 Quy trình CI/CD

8.1 Công cụ

- **GitHub Actions**: Để tự động hóa kiểm thử, xây dựng, và triển khai.
- **AWS ECS**: Để triển khai container.

8.2 Quy trình

vào nhánh main:

- Chạy kiểm thử PHPUnit cho backend.
- Chạy kiểm thử Jest cho frontend.

- Kiểm tra mã với PHP CS Fixer và ESLint.
- Xây dựng hình ảnh Docker cho backend và frontend.
- Push hình ảnh lên AWS ECR (Elastic Container Registry).
- Triển khai đến ECS.

2. Pull Request:
- Chạy kiểm thử và phân tích mã nguồn.
- Yêu cầu phê duyệt trước khi hợp nhất.

8.3 Ví dụ quy trình GitHub Actions

```
name: CI/CD Pipeline
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Backend Tests
        run: |
          docker-compose up -d
          docker exec app composer install
          docker exec app php artisan test
      - name: Frontend Tests
        run: |
          docker exec frontend npm install
          docker exec frontend npm test
  deploy:
    if: github.event_name == 'push'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Build and Push Docker Images
        run: |
          docker build -t backend ./docker/php
          docker build -t frontend ./frontend
          aws ecr get-login-password | docker login --username AWS --password-stdin <
          docker push <ecr-repo>/backend
          docker push <ecr-repo>/frontend
      - name: Deploy to ECS
        run: |
          aws ecs update-service --cluster url-shortener --service url-shortener-serv
```

9 Triển khai

9.1 Cơ sở hạ tầng

- **AWS ECS**: Lưu trữ container cho backend và frontend.
- **AWS ALB**: Định tuyến lưu lượng đến các dịch vụ ECS.
- **AWS RDS**: Cơ sở dữ liệu MySQL được quản lý.
- **AWS ElastiCache**: Cụm Redis được quản lý.
- **Route 53**: Quản lý DNS cho tên miền tùy chỉnh (ví dụ: `short.url.vn`).
- **CloudWatch**: Giám sát log và số liệu.

9.2 Các bước triển khai

1. Thiết lập tài nguyên AWS (ECS, RDS, ElastiCache, ALB, Route 53).
2. Cấu hình biến môi trường trong định nghĩa tác vụ ECS.
3. Push hình ảnh Docker lên ECR.
4. Triển khai dịch vụ bằng GitHub Actions.

10 Giám sát và bảo trì

- **Log**: Tập trung trong CloudWatch cho backend và frontend.
- **Số liệu**: Giám sát thời gian phản hồi API, tỷ lệ hit/miss của Redis, và sử dụng tài nguyên ECS.
- **Cảnh báo**: Thiết lập cảnh báo CloudWatch cho tỷ lệ lỗi cao hoặc độ trễ.
- **Sao lưu**: Snapshot RDS hàng ngày, sao lưu Redis hàng tuần.
- **Cập nhật**: Thường xuyên cập nhật Laravel, React, và các phụ thuộc.

11 Thời gian biểu

Giai đoạn	**Thời gian**	**Kết quả**
Lập kế hoạch	1 tuần	PRD, lược đồ cơ sở dữ liệu, thiết kế API
Thiết lập backend	2 tuần	Cài đặt Laravel, xác thực, API
Thiết lập frontend	2 tuần	Cài đặt React, thành phần UI
Tích hợp	1 tuần	Tích hợp frontend-backend, cache Redis
Kiểm thử	1 tuần	Kiểm thử đơn vị/tích hợp, sửa lỗi
Thiết lập CI/CD	1 tuần	Pipeline GitHub Actions
Triển khai	1 tuần	Cơ sở hạ tầng AWS, triển khai sản xuất
Tổng cộng	**9 tuần**	URL Shortener hoạt động đầy đủ

12 Rủi ro và biện pháp giảm thiểu

- **Rủi ro**: Cache Redis bị gián đoạn.
  - **Giảm thiểu**: Sử dụng ElastiCache với sao chép đa AZ.
- **Rủi ro**: Token JWT bị đánh cắp.
  - **Giảm thiểu**: Thời hạn token ngắn, lưu trữ an toàn trong cookie.
- **Rủi ro**: Truy vấn cơ sở dữ liệu chậm.
  - **Giảm thiểu**: Indexing, tối ưu truy vấn, lưu cache.
- **Rủi ro**: Triển khai thất bại.
  - **Giảm thiểu**: Chiến lược rollback trong ECS, kiểm thử CI kỹ lưỡng.

13 Tăng cường trong tương lai

- Phân tích lượt nhấp cho URL rút gọn.
- Mã ngắn tùy chỉnh cho người dùng cao cấp.
- Bảng điều khiển quản trị để quản lý người dùng và URL.
- Giới hạn tốc độ theo cấp độ người dùng.

14 Kết luận

Tài liệu PRD này cung cấp kế hoạch toàn diện để xây dựng một URL Shortener có khả năng mở rộng, an toàn và dễ bảo trì bằng Laravel và React. Bằng cách tận dụng Docker, Redis, JWT, và CI/CD, dự án sẽ mang lại sản phẩm chất lượng cao, sẵn sàng triển khai trên AWS.