

Levantamento de Requisitos - Backend Spring Boot

Projeto: Aplicação Web de Gestão de Gastos Pessoais

Data: 11/12/2025

Versão: 1.0

1. Visão Geral do Projeto

O projeto consiste em uma API REST desenvolvida com Spring Boot para gerenciar gastos pessoais de usuários. A aplicação permitirá que usuários registrem suas despesas mensais, organizem por categorias e visualizem dados para geração de gráficos de distribuição de gastos no frontend React.

2. Tecnologias e Dependências Spring Necessárias

2.1 Spring Boot Starter Web

Propósito: Desenvolvimento de APIs REST para comunicação com o frontend React.

Funcionalidades utilizadas:

- Criação de controllers REST com a anotação `@RestController`
- Mapeamento de endpoints HTTP com `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`
- Serialização e deserialização automática de JSON
- Servidor embarcado Tomcat para execução da aplicação

Endpoints que serão criados:

- `/api/auth/` - Autenticação e registro de usuários
- `/api/usuarios/` - Gerenciamento de perfil do usuário
- `/api/categorias/` - CRUD de categorias de despesas
- `/api/despesas/` - CRUD de despesas e consultas por período

2.2 Spring Boot Starter Data JPA

Propósito: Persistência de dados e mapeamento objeto-relacional.

Funcionalidades utilizadas:

- Mapeamento de entidades com @Entity, @Table, @Column
- Definição de relacionamentos com @OneToMany, @ManyToOne
- Geração automática de IDs com @GeneratedValue
- Criação de repositories com JpaRepository
- Queries customizadas com @Query e JPQL
- Controle de transações com @Transactional
- Auditoria automática com @CreationTimestamp e @UpdateTimestamp

Entidades do sistema:

- Usuario: armazena dados dos usuários cadastrados
- Categoria: armazena categorias de gastos criadas pelo usuário
- Despesa: armazena os registros de despesas com valores e datas

Relacionamentos:

- Um usuário possui muitas categorias (1:N)
- Um usuário possui muitas despesas (1:N)
- Uma categoria possui muitas despesas (1:N)

2.3 Spring Boot Starter Security

Propósito: Implementação de autenticação e autorização na aplicação.

Funcionalidades utilizadas:

- Configuração de segurança com @EnableWebSecurity
- Filtro de autenticação JWT customizado
- Criptografia de senhas com BCryptPasswordEncoder
- Proteção de endpoints por autenticação
- Configuração de políticas de sessão stateless
- Implementação de UserDetailsService para carregar dados do usuário

Regras de segurança:

- Endpoints públicos: login e registro de usuários
- Endpoints protegidos: todas as operações de categorias e despesas
- Cada usuário só pode acessar seus próprios dados

2.4 Spring Boot Starter Validation

Propósito: Validação de dados de entrada nas requisições.

Funcionalidades utilizadas:

- Validação de DTOs com @Valid
- Anotações de validação: @NotBlank, @NotNull, @Email, @Size, @DecimalMin, @PastOrPresent
- Tratamento de erros de validação com MethodArgumentNotValidException

Campos validados:

- Nome do usuário: obrigatório, entre 3 e 100 caracteres
- Email: obrigatório, formato válido, único no sistema
- Senha: obrigatória, mínimo 6 caracteres
- Nome da categoria: obrigatório, entre 2 e 50 caracteres
- Descrição da despesa: obrigatória, entre 3 e 200 caracteres
- Valor da despesa: obrigatório, maior que zero
- Data da despesa: obrigatória, não pode ser data futura

2.5 JWT (JSON Web Token)

Propósito: Autenticação stateless entre frontend e backend.

Biblioteca utilizada: jjwt (io.jsonwebtoken)

Funcionalidades implementadas:

- Geração de tokens JWT após login bem-sucedido
- Validação de tokens em cada requisição protegida
- Extração de informações do usuário a partir do token
- Configuração de tempo de expiração do token
- Assinatura de tokens com chave secreta

Fluxo de autenticação:

1. Usuário envia credenciais para o endpoint de login
2. Sistema valida credenciais e gera token JWT
3. Frontend armazena o token e envia no header Authorization
4. Filtro JWT intercepta requisições e valida o token
5. Usuário autenticado é carregado no contexto de segurança

2.6 Banco de Dados

Desenvolvimento: H2 Database (banco em memória)

- Console web habilitado para visualização dos dados
- Criação automática das tabelas a cada inicialização

- Ideal para testes e desenvolvimento local

Produção: PostgreSQL

- Banco de dados relacional robusto
- Atualização incremental das tabelas
- Persistência permanente dos dados

2.7 Lombok

Propósito: Redução de código boilerplate nas classes Java.

Anotações utilizadas:

- `@Data`: gera getters, setters, equals, hashCode e toString
- `@Builder`: implementa o padrão Builder para criação de objetos
- `@NoArgsConstructor`: gera construtor sem argumentos
- `@AllArgsConstructor`: gera construtor com todos os argumentos

3. Arquitetura e Padrões Utilizados

3.1 Estrutura de Camadas

Controller (Camada de Apresentação):

- Recebe requisições HTTP do frontend
- Valida dados de entrada
- Delega processamento para a camada de serviço
- Retorna respostas HTTP apropriadas

Service (Camada de Negócio):

- Implementa regras de negócio da aplicação
- Coordena operações entre múltiplos repositories
- Realiza validações de negócio
- Gerencia transações

Repository (Camada de Dados):

- Acesso direto ao banco de dados
- Operações CRUD básicas herdadas do `JpaRepository`
- Queries customizadas para consultas específicas

Entity (Modelo de Domínio):

- Representa as tabelas do banco de dados
- Define relacionamentos entre entidades
- Contém metadados de auditoria

DTO (Objetos de Transferência):

- Request DTOs: dados recebidos nas requisições
- Response DTOs: dados retornados nas respostas
- Separa o modelo interno da API externa

3.2 Padrão de Tratamento de Exceções

Implementação com `@RestControllerAdvice`:

- Tratamento global de exceções
- Respostas de erro padronizadas
- Exceções customizadas para recursos não encontrados
- Exceções customizadas para erros de negócio

4. Funcionalidades por Módulo

4.1 Módulo de Autenticação

Funcionalidade	Endpoint	Método	Descrição
----- ----- ----- -----			
Login	/api/auth/login	POST	Autentica usuário e retorna token JWT
Registro	/api/auth/registrar	POST	Cria novo usuário e retorna token JWT

4.2 Módulo de Usuários

Funcionalidade	Endpoint	Método	Descrição
Perfil	/api/usuarios/perfil	GET	Retorna dados do usuário logado

4.3 Módulo de Categorias

Funcionalidade	Endpoint	Método	Descrição
Listar	/api/categorias	GET	Lista todas as categorias do usuário
Buscar	/api/categorias/{id}	GET	Busca categoria por ID
Criar	/api/categorias	POST	Cria nova categoria
Atualizar	/api/categorias/{id}	PUT	Atualiza categoria existente
Deletar	/api/categorias/{id}	DELETE	Remove categoria

4.4 Módulo de Despesas

Funcionalidade	Endpoint	Método	Descrição
Listar	/api/despesas	GET	Lista todas as despesas do usuário
Listar por período	/api/despesas/periodo	GET	Lista despesas em um intervalo de datas
Buscar	/api/despesas/{id}	GET	Busca despesa por ID
Criar	/api/despesas	POST	Registra nova despesa
Atualizar	/api/despesas/{id}	PUT	Atualiza despesa existente
Deletar	/api/despesas/{id}	DELETE	Remove despesa
Total mensal	/api/despesas/total-mensal	GET	Retorna soma das despesas do mês
Por categoria	/api/despesas/por-categoria	GET	Retorna gastos agrupados por categoria

5. Modelo de Dados

5.1 Tabela: usuarios

Campo	Tipo	Restrições	Descrição
id	BIGINT	PK, AUTO_INCREMENT	Identificador único
nome	VARCHAR(100)	NOT NULL	Nome completo do usuário
email	VARCHAR(150)	NOT NULL, UNIQUE	Email para login
senha	VARCHAR(255)	NOT NULL	Senha criptografada

ativo	BOOLEAN NOT NULL, DEFAULT TRUE	Status do usuário
data_criacao	TIMESTAMP NOT NULL	Data de criação do registro
data_atualizacao	TIMESTAMP	Data da última atualização

5.2 Tabela: categorias

Campo	Tipo	Restrições	Descrição
-----	-----	-----	-----
id	BIGINT	PK, AUTO_INCREMENT	Identificador único
nome	VARCHAR(50)	NOT NULL	Nome da categoria
descricao	VARCHAR(200)		Descrição opcional
icone	VARCHAR(50)		Nome do ícone para o frontend
cor	VARCHAR(7)		Cor em formato hexadecimal
usuario_id	BIGINT	FK, NOT NULL	Referência ao usuário
data_criacao	TIMESTAMP	NOT NULL	Data de criação do registro

5.3 Tabela: despesas

Campo	Tipo	Restrições	Descrição
-----	-----	-----	-----
id	BIGINT	PK, AUTO_INCREMENT	Identificador único
descricao	VARCHAR(200)	NOT NULL	Descrição da despesa
valor	DECIMAL(10,2)	NOT NULL	Valor da despesa
data	DATE	NOT NULL	Data da despesa
observacao	VARCHAR(500)		Observações adicionais
categoria_id	BIGINT	FK, NOT NULL	Referência à categoria
usuario_id	BIGINT	FK, NOT NULL	Referência ao usuário
data_criacao	TIMESTAMP	NOT NULL	Data de criação do registro
data_atualizacao	TIMESTAMP		Data da última atualização

6. Configurações Necessárias

6.1 Propriedades da Aplicação

Propriedade	Valor	Descrição
-----	-----	-----
server.port	8080	Porta do servidor
jwt.secret	(chave secreta)	Chave para assinatura JWT
jwt.expiration	86400000	Tempo de expiração do token (24h)
spring.jpa.show-sql	true/false	Exibir SQL no console
spring.jpa.hibernate.ddl-auto	create-drop/update	Estratégia de criação de tabelas

6.2 Configuração de CORS

Origens permitidas:

- http://localhost:3000 (React com Create React App)
- http://localhost:5173 (React com Vite)

Métodos permitidos: GET, POST, PUT, DELETE, PATCH, OPTIONS

Headers permitidos: Todos

Credenciais: Habilitado

7. Anotações Spring Utilizadas

7.1 Anotações de Configuração

Anotação	Propósito
-----	-----
@SpringBootApplication	Classe principal da aplicação
@Configuration	Classe de configuração
@EnableWebSecurity	Habilita configurações de segurança
@Bean	Define um bean gerenciado pelo Spring

7.2 Anotações de Componentes

Anotação	Propósito
-----	-----
@RestController	Controller REST que retorna JSON
@Service	Componente de serviço com regras de negócio
@Repository	Componente de acesso a dados
@Component	Componente genérico gerenciado pelo Spring

7.3 Anotações de Mapeamento HTTP

Anotação	Propósito
-----	-----
@RequestMapping	Define o path base do controller
@GetMapping	Mapeia requisições GET
@PostMapping	Mapeia requisições POST

@PutMapping	Mapeia requisições PUT
@DeleteMapping	Mapeia requisições DELETE
@PathVariable	Extrai variável do path da URL
@RequestBody	Extrai corpo da requisição
@RequestParam	Extrai parâmetro da query string

7.4 Anotações JPA

Anotação	Propósito
----- -----	
@Entity	Define uma classe como entidade JPA
@Table	Configura nome da tabela
@Id	Define chave primária
@GeneratedValue	Configura geração automática de ID
@Column	Configura propriedades da coluna
@ManyToOne	Define relacionamento N:1
@OneToMany	Define relacionamento 1:N
@JoinColumn	Configura coluna de foreign key
@Transactional	Gerencia transação do método

7.5 Anotações de Validação

Anotação	Propósito
----- -----	
@Valid	Ativa validação do objeto
@NotBlank	Campo não pode ser nulo ou vazio
@NotNull	Campo não pode ser nulo
@Email	Valida formato de email
@Size	Valida tamanho mínimo e máximo
@DecimalMin	Valida valor mínimo decimal
@PastOrPresent	Data deve ser passada ou presente

8. Dependências Maven (pom.xml)

Dependência	Finalidade
----- -----	
spring-boot-starter-web	API REST
spring-boot-starter-data-jpa	Persistência de dados
spring-boot-starter-security	Autenticação e autorização
spring-boot-starter-validation	Validação de dados
postgresql	Driver PostgreSQL (produção)
h2	Banco H2 (desenvolvimento)

jjwt-api	JWT para autenticação
jjwt-impl	Implementação JWT
jjwt-jackson	Serialização JWT
lombok	Redução de boilerplate
spring-boot-starter-test	Testes unitários
spring-security-test	Testes de segurança

9. Resumo dos Conceitos Spring Aplicados

Conceito	Aplicação no Projeto
Injeção de Dependência	Autowired em services e repositories
IoC Container	Gerenciamento de beans pelo Spring
REST Controllers	Endpoints para o frontend React
Spring Data JPA	Acesso simplificado ao banco de dados
Spring Security	Proteção de endpoints e autenticação JWT
Bean Validation	Validação automática de dados de entrada
Exception Handling	Tratamento global com ControllerAdvice
Profiles	Configurações separadas para dev e prod
Transactions	Controle de transações nos services

Documento preparado para: Projeto Acadêmico - Gestão de Gastos Pessoais

Tecnologia principal: Spring Boot 3.2.0 com Java 21