



Guia de Testes - Timming LoveU



Visão Geral

Este documento fornece instruções completas sobre como executar e entender os testes automatizados implementados na aplicação Timming LoveU.



Tecnologias de Teste

- **Jest** - Framework de testes JavaScript
- **React Testing Library** - Biblioteca para testar componentes React
- **@testing-library/jest-dom** - Matchers customizados para Jest
- **@testing-library/user-event** - Simulação de interações do usuário



Estrutura de Testes

```
timming_loveu/  
├── components/  
│   ├── dashboard/  
│   │   ├── __tests__/  
│   │   │   ├── couple-info.test.tsx  
│   │   │   ├── relationship-stats.test.tsx  
│   │   │   ├── media-gallery.test.tsx  
│   │   │   ├── media-upload.test.tsx  
│   │   │   └── milestones.test.tsx  
│   │   ├── couple-info.tsx  
│   │   ├── relationship-stats.tsx  
│   │   ├── media-gallery.tsx  
│   │   ├── media-upload.tsx  
│   │   └── milestones.tsx  
│   ├── app/  
│   │   ├── api/  
│   │   │   ├── __tests__/  
│   │   │   │   ├── media-upload.test.ts  
│   │   │   │   ├── media-list.test.ts  
│   │   │   │   ├── media-delete.test.ts  
│   │   │   │   └── couple-stats.test.ts  
│   │   ├── __mocks__/  
│   │   │   ├── prisma.ts  
│   │   │   └── next-auth.ts  
│   │   ├── jest.config.js  
│   │   └── jest.setup.js
```



Scripts de Teste

Executar todos os testes

```
npm test
```

Executar testes em modo watch (desenvolvimento)

```
npm run test:watch
```

Executar testes com cobertura de código

```
npm run test:coverage
```

Executar apenas testes unitários (componentes)

```
npm run test:unit
```

Executar apenas testes de integração (APIs)

```
npm run test:integration
```



Testes Unitários - Componentes

1. CoupleInfo Component (`couple-info.test.tsx`)

Funcionalidades testadas:

- ☒ Renderização do nome do casal
- ☒ Formatação da data de início do relacionamento
- ☒ Exibição da contagem de visualizações
- ☒ Renderização da mensagem do casal (quando presente)
- ☒ Exibição do link da página pública
- ☒ Abertura da página pública em nova aba
- ☒ Compartilhamento via navigator.share (quando disponível)
- ☒ Fallback para clipboard quando share não disponível
- ☒ Renderização do banner ou ícone de coração

Exemplo de teste:

```
it('should copy link to clipboard when share button is clicked', async () => {
  render(<CoupleInfo {...mockCoupleData} />)

  const shareButton = screen.getByText('Compartilhar')
  fireEvent.click(shareButton)

  await waitFor(() => {
    expect(navigator.clipboard.writeText).toHaveBeenCalledWith(
      'http://localhost:3000/joao-maria'
    )
    expect(toast.success).toHaveBeenCalledWith('Link copiado para a área de transfer-
ência!')
  })
})
```

2. RelationshipStats Component (`relationship-stats.test.tsx`)

Funcionalidades testadas:

- ☒ Renderização da duração do relacionamento (anos, meses, dias)
- ☒ Exibição do total de dias juntos
- ☒ Estatísticas de mídia (total, imagens, vídeos)
- ☒ Atualização do tempo em tempo real (live timer)
- ☒ Tratamento de valores zero

3. MediaGallery Component (`media-gallery.test.tsx`)

Funcionalidades testadas:

- ☒ Estado de carregamento inicial
- ☒ Carregamento e exibição de itens de mídia
- ☒ Filtros de mídia (todos, imagens, vídeos)
- ☒ Diálogo de confirmação de exclusão
- ☒ Exclusão de mídia
- ☒ Tratamento de erros de fetch
- ☒ Atualização quando refreshTrigger muda
- ☒ Estado vazio (sem mídia)
- ☒ Exibição de data do evento

4. MediaUpload Component (`media-upload.test.tsx`)

Funcionalidades testadas:

- ☒ Renderização do formulário de upload
- ☒ Seleção de arquivo
- ☒ Preview de imagem selecionada
- ☒ Entrada de título e descrição
- ☒ Envio do formulário
- ☒ Validação de arquivo obrigatório
- ☒ Validação de tamanho de arquivo
- ☒ Validação de tipo de arquivo
- ☒ Tratamento de erro de upload
- ☒ Reset do formulário após sucesso
- ☒ Desabilitação do botão durante upload

5. Milestones Component (`milestones.test.tsx`)

Funcionalidades testadas:

- ☒ Renderização de todos os marcos
- ☒ Exibição de descrições dos marcos
- ☒ Formatação de datas
- ☒ Status de marcos completados
- ☒ Status de marcos pendentes
- ☒ Estado vazio (sem marcos)
- ☒ Ordenação de marcos
- ☒ Tratamento de marco único

Testes de Integração - APIs

1. Media Upload API (`media-upload.test.ts`)

Cenários testados:

- ☒ Retorno 401 se usuário não autenticado
- ☒ Retorno 404 se usuário não encontrado no banco
- ☒ Retorno 400 se usuário não possui página de casal
- ☒ Retorno 400 se nenhum arquivo fornecido
- ☒ Retorno 400 para tipo de arquivo inválido
- ☒ Retorno 400 para arquivo muito grande (imagem > 10MB)
- ☒ Upload bem-sucedido de imagem
- ☒ Upload bem-sucedido de vídeo
- ☒ Upload com data do evento
- ☒ Tratamento de erros do servidor

Limites de tamanho:

- Imagens: máximo 10MB
- Vídeos: máximo 50MB

Tipos de arquivo aceitos:

- Imagens: JPEG, JPG, PNG, GIF, WebP
- Vídeos: MP4, WebM, QuickTime, AVI

2. Media List API (`media-list.test.ts`)

Cenários testados:

- ☒ Retorno 401 se não autenticado
- ☒ Retorno 404 se usuário não encontrado
- ☒ Retorno 400 se sem página de casal
- ☒ Listagem bem-sucedida de mídias
- ☒ Array vazio quando sem mídias
- ☒ Filtragem de apenas mídias ativas
- ☒ Tratamento de erros do servidor

3. Media Delete API (`media-delete.test.ts`)







Cenários testados:

- ☒ Retorno 401 se não autenticado
- ☒ Retorno 400 se ID não fornecido
- ☒ Retorno 404 se mídia não encontrada
- ☒ Exclusão soft delete bem-sucedida
- ☒ Tratamento de mídia já excluída
- ☒ Tratamento de erros do servidor

4. Couple Stats API (`couple-stats.test.ts`)

Cenários testados:

- ☒ Retorno 401 se não autenticado
- ☒ Retorno 404 se usuário não encontrado
- ☒ Retorno 400 se sem página de casal
- ☒ Cálculo correto da duração do relacionamento
- ☒ Estatísticas corretas de mídia
- ☒ Informações da página do casal

-  Contagem de visualizações
-  Geração de marcos (milestones)
-  Identificação de marcos completados
-  Tratamento de zero mídias
-  Cálculo para relacionamentos recentes
-  Tratamento de erros do servidor

Configuração do Jest

`jest.config.js`

Configurações principais:

- Usa `next/jest` para integração com Next.js
- Ambiente de teste: `jest-environment-jsdom`
- Setup file: `jest.setup.js`
- Mapeamento de módulos: `@/` para raiz do projeto
- Cobertura de código configurada

`jest.setup.js`

Configurações de setup:

- Mock do Next.js router (`next/navigation`)
- Mock do componente Image do Next.js
- Mock do `window.matchMedia`
- Mock do `IntersectionObserver`
- Mock do `window.location`
- Supressão de `console.error/warn` em testes

Mocks

`__mocks__/prisma.ts`

Mock do Prisma Client com funções jest para:

- user (findUnique, findMany, create, update, delete)
- couplePage (findUnique, findMany, create, update, delete)
- media (findUnique, findMany, create, update, delete)
- session (findUnique, create, delete)

`__mocks__/next-auth.ts`

Mock do NextAuth com:

- `mockSession` - sessão de usuário de teste
- `getServerSession` - função mockada
- `authOptions` - opções de autenticação

Cobertura de Código

Para visualizar o relatório de cobertura:

```
npm run test:coverage
```

O relatório será gerado em `coverage/lcov-report/index.html`

Metas de Cobertura

- **Statements:** > 80%
- **Branches:** > 75%
- **Functions:** > 80%
- **Lines:** > 80%



Debugging de Testes

Executar um teste específico

```
npm test -- couple-info.test.tsx
```

Executar testes com mais detalhes

```
npm test -- --verbose
```

Ver apenas testes falhados

```
npm test -- --onlyFailures
```



Boas Práticas

1. Estrutura de Teste AAA

- **Arrange:** Prepare o ambiente de teste
- **Act:** Execute a ação sendo testada
- **Assert:** Verifique o resultado

```
it('should render couple name correctly', () => {  
  // Arrange  
  const mockData = { /* ... */ }  
  
  // Act  
  render(<CoupleInfo {...mockData} />)  
  
  // Assert  
  expect(screen.getByText('João & Maria')).toBeInTheDocument()  
})
```

2. Usar waitFor para código assíncrono

```
await waitFor(() => {  
  expect(screen.getByText('Success')).toBeInTheDocument()  
})
```

3. Limpar mocks entre testes

```
beforeEach(() => {  
  jest.clearAllMocks()  
})
```

4. Testar comportamento, não implementação

❌ Ruim:

```
expect(component.state.value).toBe(5)
```

✅ Bom:

```
expect(screen.getByDisplayValue('5')).toBeInTheDocument()
```



Troubleshooting

Erro: “Cannot find module”

Verifique o `moduleNameMapper` no `jest.config.js` e certifique-se de que os aliases estão corretos.

Erro: “window is not defined”

Certifique-se de usar `testEnvironment: 'jest-environment-jsdom'` no `jest.config.js`.

Testes assíncronos falhando

Use `waitFor`, `findBy` queries, ou `async/await` apropriadamente.

Mock não funcionando

Verifique se o mock está no lugar correto (`__mocks__` /) ou se está sendo chamado antes do import.



Recursos Adicionais

- [Jest Documentation](https://jestjs.io/docs/getting-started) (https://jestjs.io/docs/getting-started)
- [React Testing Library](https://testing-library.com/docs/react-testing-library/intro/) (https://testing-library.com/docs/react-testing-library/intro/)
- [Testing Library Cheatsheet](https://testing-library.com/docs/react-testing-library/cheatsheet) (https://testing-library.com/docs/react-testing-library/cheatsheet)
- [Common Testing Mistakes](https://kentcdodds.com/blog/common-mistakes-with-react-testing-library) (https://kentcdodds.com/blog/common-mistakes-with-react-testing-library)

Exemplos de Uso

Testando Interação do Usuário

```
import { render, screen, fireEvent } from '@testing-library/react'
import userEvent from '@testing-library/user-event'

it('should handle user input', async () => {
  render(<MyForm />)

  const input = screen.getByLabelText('Nome')
  await userEvent.type(input, 'João')

  expect(input).toHaveValue('João')
})
```

Testando API com Mock

```
it('should fetch data successfully', async () => {
  global.fetch = jest.fn(() =>
    Promise.resolve({
      ok: true,
      json: () => Promise.resolve({ data: 'test' }),
    })
  )

  render(<MyComponent />)

  await waitFor(() => {
    expect(screen.getByText('test')).toBeInTheDocument()
  })
})
```

Checklist de Testes

Ao adicionar novos testes, verifique:

- ☐ Teste passou localmente
- ☐ Cobertura de código adequada
- ☐ Testa comportamento do usuário, não implementação
- ☐ Usa queries semânticas (getByRole, getByLabelText)
- ☐ Limpa mocks e side effects
- ☐ Trata código assíncrono apropriadamente
- ☐ Nomes de teste descritivos e claros
- ☐ Segue padrão AAA (Arrange, Act, Assert)

Suporte

Se encontrar problemas ou tiver dúvidas sobre os testes, consulte:

1. Esta documentação
2. Logs de erro detalhados
3. Documentação oficial das ferramentas
4. Issues similares no GitHub do projeto

Última atualização: Outubro 2024

Versão: 1.0.0