Problemas gerais:

1. Rotas retornando código 200 mesmo quando um erro ocorre.
2. Código de erro incorreto pra algumas exceções
3. Em algumas requisições, quando um campo é informado fora da validação esperada, é lançado um erro 404, mas não é informado o motivo.
4. Não é validado se o usuário informado nas rotas pertence a empresa do usuário logado. Seria interessante criar um middleware pra validar isso, visto que diversas rotas precisam do usuário.
5. criar scopes de where de empresa nos repositórios

GET api/user/{id}:

1. Não há validação pra verificar se o id informado na url pertence a empresa do usuário logado. O mesmo vale pra rota de atualizar usuário. Ao realizar uma ação no banco de dados irá dar erro, que não é tratado pela aplicação.

POST api/users/{id}/account:

1. ativação da conta não verifica se usuário está ativo

api/user/register:

1. UseCases/User/CreateFirstUser: várias responsabilidades ->Criação da empresa, validação do usuário, criação do usuário, criação do token

api/users/{id}/account:

1. UseCases/Account/Register.php: várias responsabilidades -> buscando usuário, criando conta no serviço externo, registrando conta no banco local

api/users/{id}/card:

1. UseCases/Card/Register.php: várias responsabilidades -> validando registro, registrando cartão no serviço externo, salvando no banco local

api/users/{id}:

1. UseCases/User/Update.php: validando usuário, atualizando usuário, gerenciando senha

Problemas comuns a diversas classes:

- o método handle de todos os useCases tá retornando o resultado da ação fora do bloco de try catch. Colocando o retorno dentro do bloco, evita que aconteçam comportamentos inesperados.

Problemas por classe:

Model/Card.php: método user não condiz com o relacionamento

UserController.php

- injetar as dependências da controller ao invés de dar new em tudo.
- centralizar o Auth::user()->company_id em um método pra evitar repetir esse trecho de código nos métodos.
- alguns nomes de métodos: show e index. Eu colocaria um nome que explicitar mais ao que está sendo feito. Por exemplo: getUserDetails, listUsers.
- UseCases/User/show.php -> nome da classe começando com letra minúscula.

UseCases/User/CreateFirstUser.php:

- $token, $user, $company -> variáveis com visibilidade protected, mas são usadas apenas na classe

UseCases/User/show.php:

- atributos do construtor informados como $a e $b. Renomear pra ficar claro que esses valores são referentes ao id do usuário e id da empresa.

UseCases/Login.php:

1. new create_token($this->id) : alias de classe com padrão incorreto.

Falta de tipagem:

1. app\UseCases\User\CreateFirstUser.php tipar retorno do handler
2. app\UseCases\User\Create.php tipar retorno do handler
3. app\UseCases\User\Login.php tipar retorno do handler
4. app\UseCases\User\Update.php tipar retorno do handler
5. SanitizesInput falta tipagem do parâmetro do método sanitize

Testes:

1. tests\Feature\Company\UpdateTest teste com erro

Account/RegisterTest:

1. Testar quando usuário não tem permissão
2. Testar quando usuário não existe
3. Testar falha na integração com banking
4. Testar quando conta já existe

Account/ShowTest:

1. Testar quando usuário não tem permissão
2. Testar quando conta não existe
3. Testar falha na integração com banking

Account/ActiveTest:

1. Adicionar teste de não autorização, tipo testActiveWhenUnauthorized, pra um usuário que não tenha permissão
2. Adicionar um teste pra quando a conta não for encontrada
3. Adicionar um teste pra erro na integração com banking
4. Adicionar um teste de ativar uma conta já ativa

Account/BlockedTest:

1. Adicionar teste de bloqueio quando usuário não tem permissão
2. Adicionar teste de conta não encontrada
3. Adicionar teste de erro na integração com banking
4. Adicionar teste de conta já bloqueada

Card/RegisterTest:

1. Testar quando usuário não tem permissão
2. Testar quando conta não existe
3. Testar falha na integração com banking
4. Testar quando cartão já existe
5. Testar PIN inválido

Card/ShowTest:

1. Testar quando usuário não tem permissão
2. Testar quando cartão não existe
3. Testar falha na integração com banking
4. Verificar estrutura da resposta

User/RegisterTest:

1. Testar CPF inválido
2. Testar CNPJ inválido
3. Testar email inválido
4. Testar email já cadastrado
5. Testar CPF já cadastrado
6. Testar CNPJ já cadastrado

User/CreateTest:

1. Testar CPF inválido
2. Testar email inválido
3. Testar email já cadastrado
4. Testar CPF já cadastrado
5. Testar tipo de usuário inválido

User/LoginTest:

1. Testar email não encontrado
2. Testar senha incorreta
3. Testar conta bloqueada
4. Verificar token retornado

User/IndexTest:

1. Testar paginação

User/UpdateTest:

1. Testar validação de campos
2. Testar email inválido
3. Testar email já cadastrado
4. Testar tipo inválido
5. Testar senha inválida

Company/UpdateTest:

1. Testar CNPJ inválido
2. Testar CNPJ já cadastrado

Company/ShowTest:

1. Testar quando usuário não tem permissão
2. Testar quando empresa não existe

Observações/Sugestões:

1. Adicionar as policies criadas no service provider AuthServiceProvider
2. Para os problemas de UseCases com muitas responsabilidades, sugiro a criação de um orquestrador de atividades. Aqui na empresa chamamos de Workflow, que é basicamente uma classe que orquestra a execução de algumas outras, por ordem. Essas classes serão activities, onde cada uma será responsável por apenas uma coisa, seguindo o princípio da responsabilidade única.
3. Capturar as exceções lançadas pelas policies.
4. Resources: tipar o parâmetro e o retorno de todos os toArray de todos os resources
5. Não existe rota de deletar usuários ou empresas. Não deveria existir?
6. trait SanitizesInput não está sendo utilizada pra nada.
7. seria interessante adicionar cache em algumas consultas
8. adicionar logs de integração com api externa
9. armazenar dados como os logs e tokens de usuário em algum banco NoSql.