```
class FeedbackRoute(models.Model):
```

Testes contra erros para a classe *FeedbackRoute*, levando em conta como usuários podem interagir:

- **rating fora do intervalo permitido.**

rating aceita apenas valores inteiros de 0 a 5, então usuários podem tentar eviar valores como 6, -3, 18, ou até strings.

- **user vazio ou muito longo.**

user tem limite de 100 caracteres e não lida com null ou string vazia.
Assim, o usuário pode tentar enviar nome com mais de 100 caracteres ou campo vazio.
Causando assim falha na validação com erro de integridade ou de formulário.

- **message em branco.**

Por padrão, *TextField* permite vazio, mas se o front ou lógica exigir texto, isso pode causar UX ruim.
Assim, com uma entrada vazia, pode ser gerado um feedback sem conteúdo útil.
É sugerível adicionar *blank=False* se quiser forçar uma mensagem.

- **Inserção manual de votos**

Em *upvotes* e *downvotes* existe *default=0*, mas podem ser modificáveis pelo usuário, dependendo da implementação.
Um usuário pode por exemplo editar os votos e causar um transtorno.
Se alterar esse campo para read-only no forms ou API, é possível evitar o problema.

- **Data manual**

Se alguém tenta enviar *created_at*: "30/02/1999", deveria ser inválido.
É necessário verificar as datas que são inseridas.

- **caracteres especiais**

Nomes com acentos, emojis ou caracteres especiais podem ser mal interpretados dependendo da base do frontend.
É necessário verificar se o encoding está correto.

```
class Route(models.Model):
```

Testes contra erros para a classe *Route*, levando em conta como usuários podem interagir:

- **Velocidade média inválida em *estimated_time***

Se *average_speed* for “0”, negativo ou um valor inválido tipo string, emoji e etc, a função lança um *ValueError*.

Poderia retornar *None* ou *float("inf")*, dependendo da aplicação, ou validar antes de chamar.

- **distance negativa ou inválida**

distance aceita qualquer float, inclusive negativos ou “0”.
Rotas com 0km, ou -1km não fazem sentido, logo adicionar um valor mínimo é ideal:
validators=[MinValueValidator(0.01)]

- **Relacionamento com feedbacks vazios ou inconsistentes**

feedbacks pode ser deixado vazio pelo usuário, e o *show_feedbacks* falha ou imprime um lixo do cachê.
É necessário verificar se existem feedbacks, com por exemplo:

```
if not self.feedbacks.exists():
    print("Nenhum feedback disponível.")
```

- **Quantidade negativa em *show_feedbacks(quantity)***

Já que *quantity* não tem verificação de valor, o usuário pode passar um número negativo, resultando em um corte da string com [:-1]

Um exemplo de código:

```
if quantity <= 0:
    print("Quantidade deve ser maior que zero.")
    return
```

- **Busca por localização nula ou genérica em “feedbacks_by_location”**

A função espera uma string de localização, porém se um usuário passa uma string vazia ou muito grande, pode acabar pegando todas as rotas.

É possível verificar o input com:

```
if not location.strip():
    print("Localização inválida.")
    return
```

- **Performance com prefetch_related**

O método *feedbacks_by_location* usa *prefetch_related*, que pode bugar e causar lentidão se houver muitas rotas e muitos feedbacks.
Seria legal limitar a quantidade de feedbacks por rota.

- **Falha em *create_route***

Já que cria rota e salva rotas, mas não valida se os dados são fazem sentido, pode existir criação de rotas que o ponto de partida é igual ao de chegada (*start_point == end_point*), ou com a distância sendo igual a zero.