Sistema de biblioteca

TRABALHO DE BACK-END FRAMEWORKS

Membros

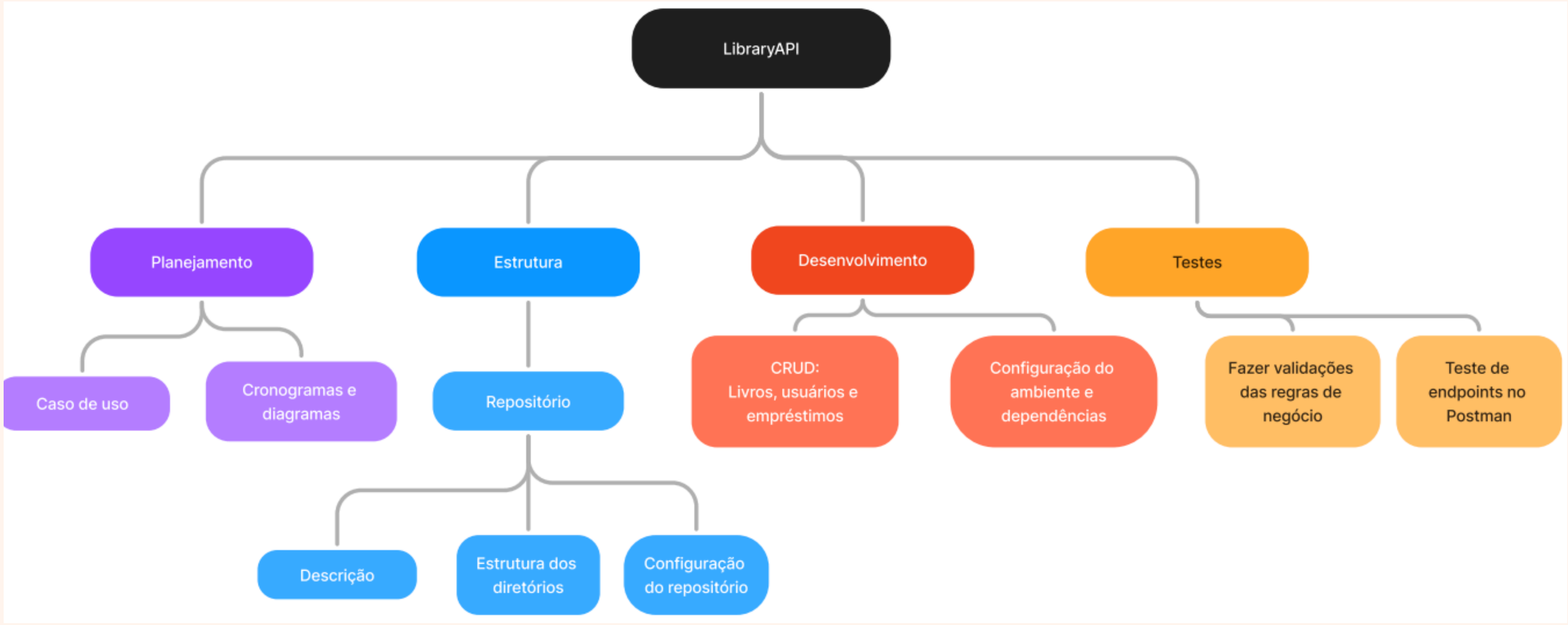
Aline Bianca Arantes da Silva - 01617756
• Documentadora e Desenvolvedora

Emerson Wallace Barcelos de Araújo - 01609132
• Gerente de configuração e Desenvolvedor

Maria Karolina Rodrigues de Andrade - 01631286
• Documentadora e Scrum Master

Sumário

Estrutura Analítica do Projeto



Custos e Investimentos

NOME	Maria Karolina Rodrigues de Andrade		
CARGO	Scrum Master		
VALOR POR HORA	R$ 56,00	*Inserir um valor médio*	
TOTAL DE HORAS NO MÊS	160 Horas	*Horas mensais dedicadas*	
SALÁRIO BASE	R$ 8.960,00	*Salário base do funcionário*	
IMPOSTOS TOTAIS (% MÉDIO)	20%	*Média dos impostos sobre salário*	
IMPOSTOS TOTAIS A PAGAR	R$ 1.792,00	*Valor dos impostos sobre salário*	
BENEFÍCIOS (VR, VT, SAÚDE)	R$ 896,00	*Benefícios ao funcionário*	
APROVISIONAMENTOS	R$ 139,78	*Processos, seguros, despesas*	
CUSTO TOTAL MENSAL	**R$ 11.787,78**	*Custo total para a empresa*	

NOME	Emerson Wallace Barcelos de Araújo		
CARGO	Desenvolvedor		
VALOR POR HORA	R$ 35,00	*Inserir um valor médio*	
TOTAL DE HORAS NO MÊS	160 Horas	*Horas mensais dedicadas*	
SALÁRIO BASE	R$ 5.600,00	*Salário base do funcionário*	
IMPOSTOS TOTAIS (% MÉDIO)	20%	*Média dos impostos sobre salário*	
IMPOSTOS TOTAIS A PAGAR	R$ 1.120,00	*Valor dos impostos sobre salário*	
BENEFÍCIOS (VR, VT, SAÚDE)	R$ 560,00	*Benefícios ao funcionário*	
APROVISIONAMENTOS	R$ 87,36	*Processos, seguros, despesas*	
CUSTO TOTAL MENSAL	**R$ 7.367,36**	*Custo total para a empresa*	

NOME	Aline Bianca Arantes da Silva		
CARGO	Desenvolvedora		
VALOR POR HORA	R$ 35,00	*Inserir um valor médio*	
TOTAL DE HORAS NO MÊS	160 Horas	*Horas mensais dedicadas*	
SALÁRIO BASE	R$ 5.600,00	*Salário base do funcionário*	
IMPOSTOS TOTAIS (% MÉDIO)	20%	*Média dos impostos sobre salário*	
IMPOSTOS TOTAIS A PAGAR	R$ 1.120,00	*Valor dos impostos sobre salário*	
BENEFÍCIOS (VR, VT, SAÚDE)	R$ 560,00	*Benefícios ao funcionário*	
APROVISIONAMENTOS	R$ 87,36	*Processos, seguros, despesas*	
CUSTO TOTAL MENSAL	**R$ 7.367,36**	*Custo total para a empresa*	

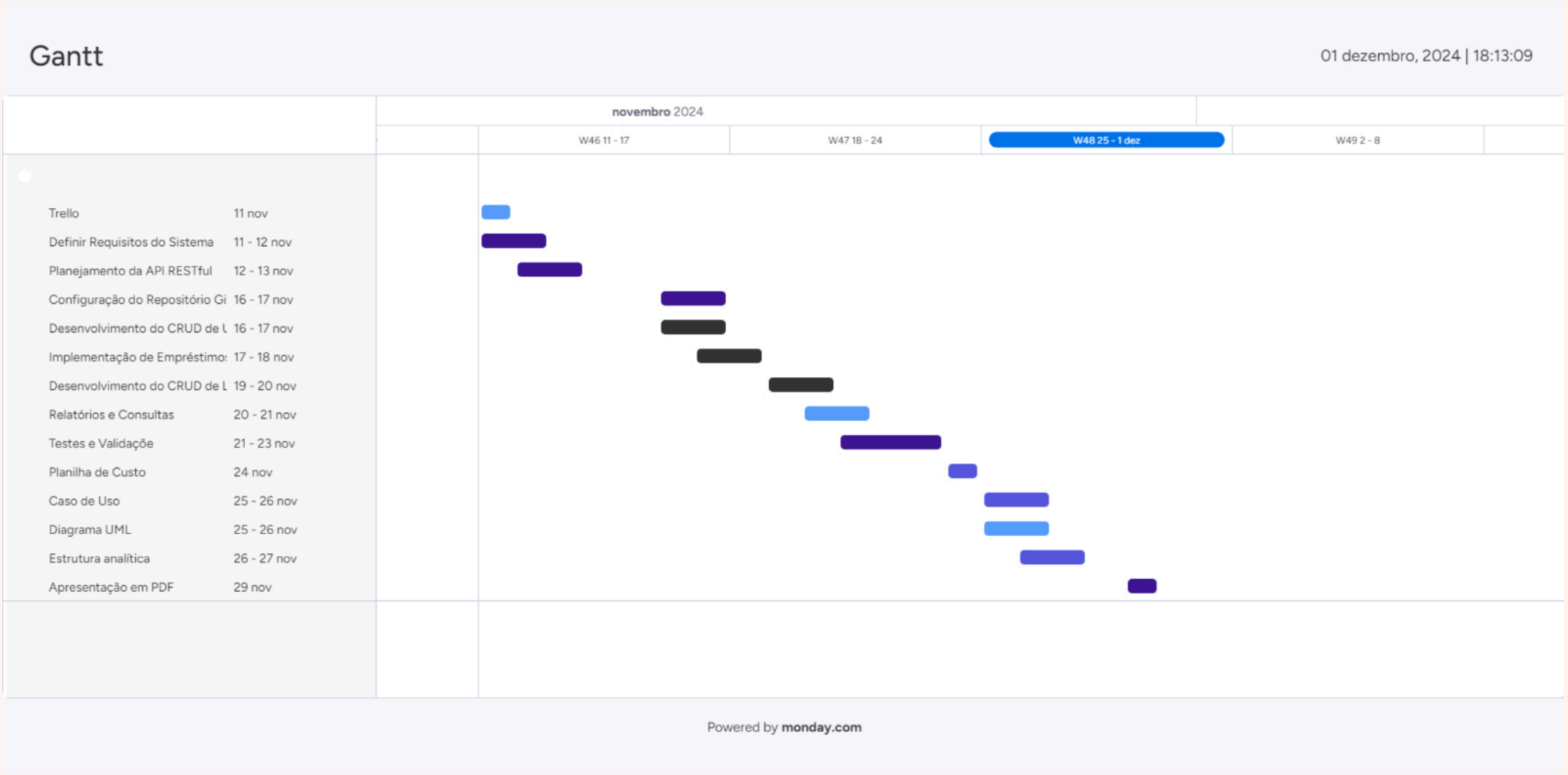
Custos e Investimentos

LICENÇAS DE SOFTWARE			
Microsoft Visual Studio	R$	375,00	*Valor mensal*
MySQL Enterprise	R$	420,00	*Valor mensal*
Postman	R$	350,00	*Valor mensal*
Figma Premium	R$	1.350,00	*Valor mensal*
CUSTO TOTAL MENSAL	**R$**	**2.495,00**	*Custo total para a empresa*

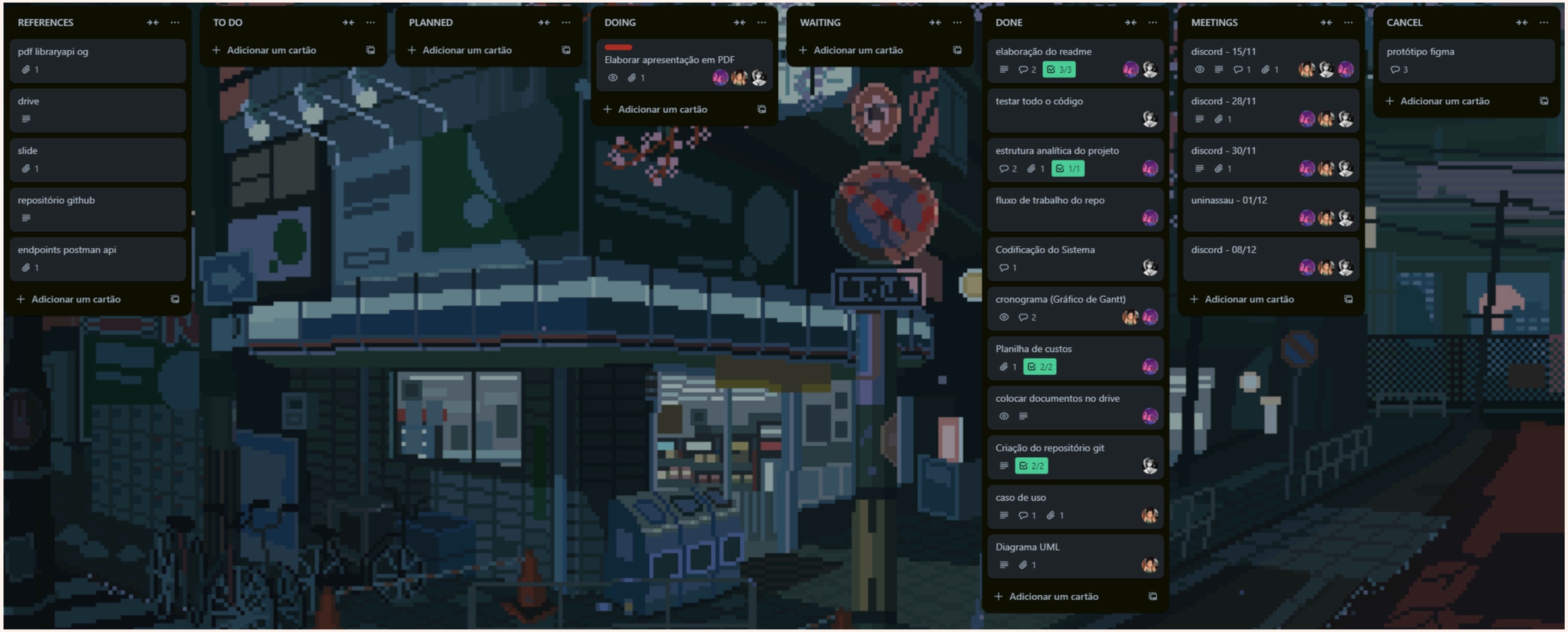
ASSINATURA DE SERVIÇOS			
Udemy Business	R$	1.000,00	*Valor mensal*
GitHub Enterprise	R$	576,00	*Valor mensal*
Amazon Web Services	R$	3.000,00	*Valor mensal*
Trello	R$	86,00	*Valor mensal*
Office 365	R$	105,00	*Valor mensal*
CUSTO TOTAL MENSAL	**R$**	**4.767,00**	*Custo total para a empresa*

INFRAESTRUTURA E HARDWARE			
Link Internet 1Gbps	R$	500,00	*Valor mensal*
Roteador TP-Link	R$	300,00	*Valor fixo*
Notebooks i7	R$	7.000,00	*Valor fixo*
Mesas de escritório	R$	1.200,00	*Valor fixo*
Cadeiras de escritório	R$	2.400,00	*Valor fixo*
CUSTO TOTAL MENSAL	**R$**	**11.400,00**	*Custo total para a empresa*

Cronograma de Gantt



Trello


REFERENCES
pdf libraryapi og
1
drive
slide
1
repositório github
endpoints postman api
1
Adicionar um cartão
TO DO
Adicionar um cartão
PLANNED
Adicionar um cartão
DOING
Elaborar apresentação em PDF
1
Adicionar um cartão
WAITING
Adicionar um cartão
DONE
elaboração do readme
2
3/3
testar todo o código
estrutura analítica do projeto
2
1
1/1
fluxo de trabalho do repo
Codificação do Sistema
1
cronograma (Gráfico de Gantt)
2
Planilha de custos
1
2/2
colocar documentos no drive
Criação do repositório git
2/2
caso de uso
1
1
Diagrama UML
1
Adicionar um cartão
MEETINGS
discord - 15/11
1
1
discord - 28/11
1
discord - 30/11
1
uninassau - 01/12
discord - 08/12
Adicionar um cartão
CANCEL
protótipo figma
3
Adicionar um cartão

Casos de Uso

Recuperar Senha

ID	UC002
Título	Recuperar senha
Ator principal	Usuário
Objetivo	Permitir que o usuário redefina a senha de acesso ao sistema caso a tenha esquecido ou perdido.
Pré-condição	O usuário deve ter um e-mail válido previamente cadastrado no sistema.

Gerenciamento de Livros

ID	UC003
Título	Gerenciamento de livros
Ator principal	Usuário
Objetivo	Permitir que usuários visualize a lista de livros disponíveis.
Pré-condição	Usuários devem estar cadastrados e autenticados para acessar funcionalidades restritas.
Pós-condição	O usuário consegue visualizar a lista de livros disponíveis ou os detalhes de um livro específico.

Registro de Empréstimos e Devoluções

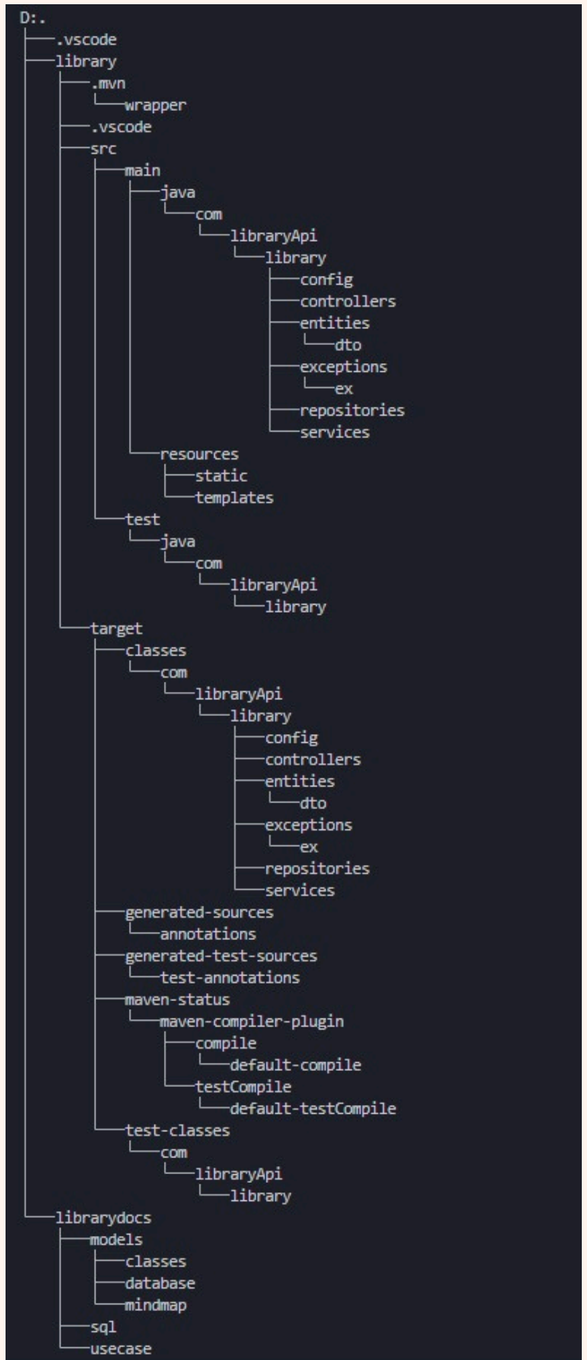
ID	UC004
Título	Empréstimos e Devoluções
Ator principal	Usuário
Objetivo	Permitir que o usuário solicite o empréstimo de livros disponíveis no acervo, garantindo o controle de prazos e limites.
Pré-condição	O usuário deve estar autenticado no sistema, o livro solicitado deve estar disponível no acervo, e o usuário não pode ter excedido o limite máximo de empréstimos ativos permitido.
Pós-condição	O empréstimo do livro é registrado com sucesso, incluindo a data prevista para devolução.

Cadastro e Login

ID	UC001
Título	Login e cadastro
Ator principal	Usuário
Objetivo	Permitir que usuários se cadastrem, realizem login, acessem livros, façam empréstimos e gerenciem suas ações no sistema.
Pré-condição	Usuários devem estar cadastrados e autenticados para acessar funcionalidades restritas.
Pós-condição	Usuários autenticados podem acessar os livros, registrar empréstimos e devoluções

Fluxo de trabalho do repositório

Work tree


D:.
.vscode
library
.mvn
wrapper
.vscode
src
main
java
com
libraryApi
library
config
controllers
entities
dto
exceptions
ex
repositories
services
resources
static
templates
test
java
com
libraryApi
library
target
classes
com
libraryApi
library
config
controllers
entities
dto
exceptions
ex
repositories
services
generated-sources
annotations
generated-test-sources
test-annotations
maven-status
maven-compiler-plugin
compile
default-compile
testCompile
default-testCompile
test-classes
com
libraryApi
library
librarydocs
models
classes
database
mindmap
sql
usecase

Extratos do código

(Partes mais importantes)

Classes dos Rest Controllers

```java
@RestController
@RequestMapping("/books")
@RequiredArgsConstructor
public class BookController {

    private final BooksService booksService;

    @PostMapping("/create")
    public ResponseEntity<Books> createNewBook(@RequestBody Books booksDto){
    return ResponseEntity.status(201).body(booksService.addNewBook(booksDto));

    }

    @GetMapping
    public ResponseEntity<List<Books>> findAllBooks(){
        return ResponseEntity.ok(booksService.findAllBooks());
    }


    @GetMapping("/{id}")
    public ResponseEntity<Books> finBook(@PathVariable Long id){
        return ResponseEntity.ok(booksService.findBook(id));
    }


    @PatchMapping("/update/{id}")
    public ResponseEntity<Books> updateBook(@PathVariable Long id, @RequestBody Books booksDto){
       return ResponseEntity.ok(booksService.updateBook(id, booksDto));

    }

    @DeleteMapping("/delete/{id}")
    public ResponseEntity<Void> deleteBook(@PathVariable Long id){
        booksService.deleteBook(id);
        return ResponseEntity.ok().build();

    }

    @GetMapping("/mostBorrowed")
    public ResponseEntity<List<Books>> findMostBorrowedBooks(){
        return ResponseEntity.ok(booksService.mostBorroweBooks());
    }

}

```

```java
@RestController
@RequestMapping("/user")
@RequiredArgsConstructor
public class UserController {

    private final UserService userService;

    @PostMapping("/create")
    public ResponseEntity<User> createNewUser(@Valid @RequestBody User user){
        return ResponseEntity.status(201).body(userService.addUser(user));
    }

    @GetMapping
    public ResponseEntity<List<User>> findAllUsers(){
        return ResponseEntity.ok(userService.findAllUsers());
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> findUser(@PathVariable Long id){
        return ResponseEntity.ok(userService.findUser(id));
    }

    @PatchMapping("/update/{id}")
    public ResponseEntity<User> updateUser(@PathVariable Long id, @RequestBody @Valid User  user){
       return ResponseEntity.ok(userService.updateUser(id,user));

    }

    @DeleteMapping("/delete/{id}")
    public ResponseEntity<Void> deleteUser(@PathVariable Long id){
        userService.deleteUser(id);
        return ResponseEntity.ok().build();

    }
}

```

Classes dos Rest Controllers

```java
@RestController
@RequestMapping("/borrows")
@RequiredArgsConstructor
public class BorrowedBooksController {

    private final BorrowedBooksService borrowedBooksService;

    @PostMapping("/newBorrow")
    public ResponseEntity<BorrowedBooks> borrowNewBook(@RequestBody BorrowedBooksDto borrowedBooksDto) {
        return ResponseEntity.status(201)
                .body(borrowedBooksService.borrowNewBook(
                    borrowedBooksDto.userId(),
                    borrowedBooksDto.bookId(),
                    borrowedBooksDto.returnDate()));
    }

    @GetMapping
    public ResponseEntity<List<BorrowedBooks>> findAllBorrowedBooks(){
        return ResponseEntity.ok(borrowedBooksService.findAllBorrowedBooks());
    }

    @DeleteMapping("/return/{userId}/{bookId}")
    public ResponseEntity<Void> returnBook(@PathVariable Long userId, @PathVariable  Long bookId ){
        borrowedBooksService.returnBook(userId, bookId);
        return ResponseEntity.ok().build();
    }
}
```

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/PendingBooks")
public class PendingBooksController {

    private final PendingBooksService pendingBooksService;

    @PostMapping("/NewPending")
    public ResponseEntity<PendingBooks> newPending(@RequestBody PendingBooksDto pendingBooksDto){

        return ResponseEntity.status(201).body(pendingBooksService.newPendingBook(pendingBooksDto.userId(),
                pendingBooksDto.bookId()));
    }

    @GetMapping
    public ResponseEntity<List<PendingBooks>> findAllPendingBooks(){
        return ResponseEntity.ok(pendingBooksService.findAllPendingBooks());
    }

    @DeleteMapping("/returnPending/{userId}/{bookId}")
    public ResponseEntity<Void> returnBook(@PathVariable Long userId, @PathVariable  Long bookId ){
        pendingBooksService.returnPendingBook(userId, bookId);
        return ResponseEntity.ok().build();
    }
}
```

Classes dos Serviços

```java
@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository userRepository;

    public User addUser(User user){
        return userRepository.save(user);
    }

    public List<User> findAllUsers(){
        return userRepository.findAll();
    }

    public User findUser(Long id){

        return userRepository.findById(id).orElseThrow(
            () -> new ContentNotFound("Id not Found")
        );
    }

    public User updateUser(Long id, User user){
        User userToUpdate = this.findUser(id);

        if (user.getEmail()!=null){
            userToUpdate.setEmail(user.getEmail());
        }
        if (user.getAddress() != null){
            userToUpdate.setAddress(user.getAddress());
        }
        if (user.getName() != null){
            userToUpdate.setName(user.getName());
        }
        if (user.getNumber()!= null) {
            userToUpdate.setNumber(user.getNumber());
        }

        return userRepository.save(userToUpdate);

    }

    public void deleteUser(Long id){
        var user = this.findUser(id);
        userRepository.delete(user);

            try {
            userRepository.delete(user);
        }
        catch (DataIntegrityViolationException e){

            throw new UserAlreadyBorrowed("User cannot be deleted with borrowed books!");
        }

    }
}
```

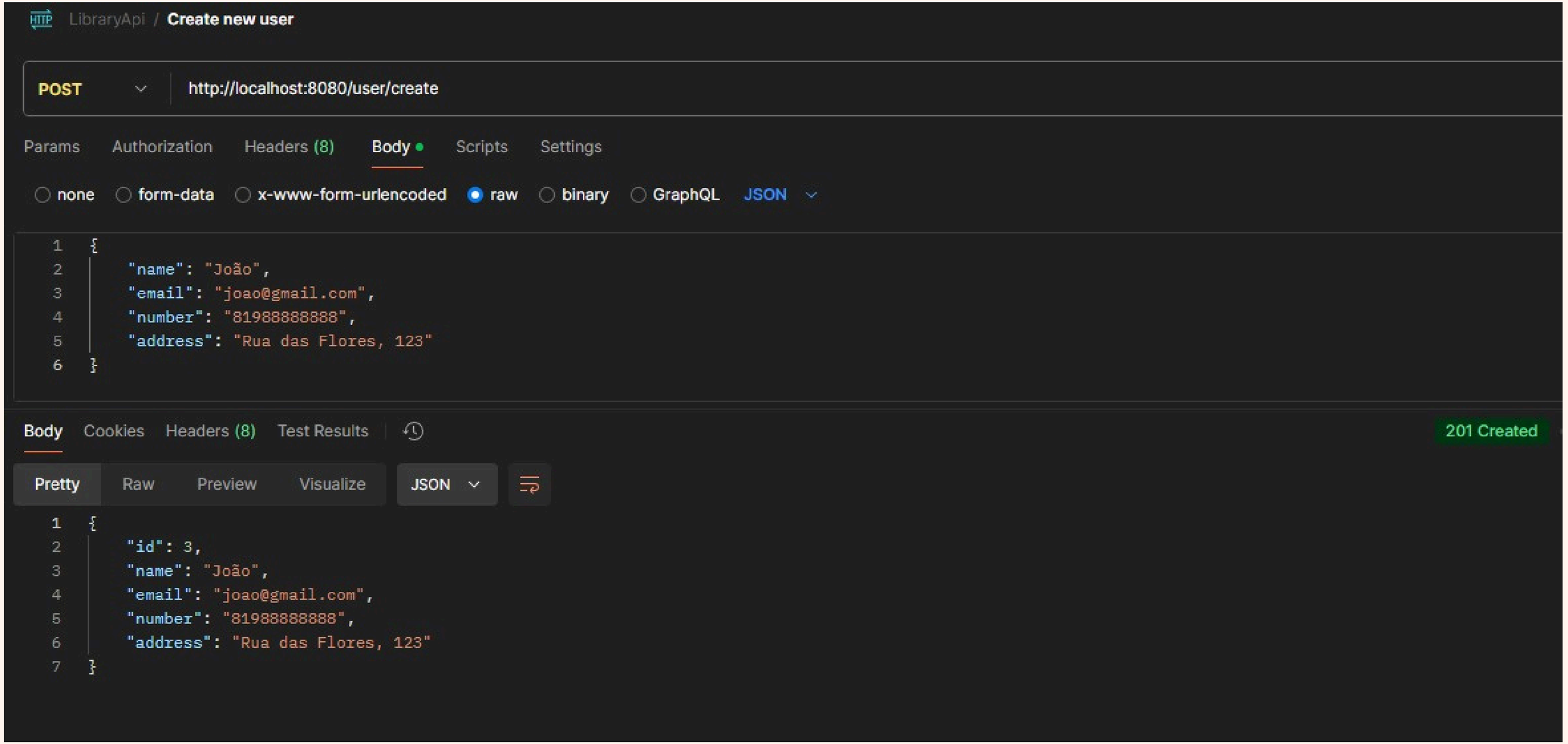
```java
@Service
@RequiredArgsConstructor
public class BooksService {

    private final BooksRepository booksRepository;

    public Books addNewBook(Books books){
        return booksRepository.save(books);
    }

    public List<Books> findAllBooks(){
        return booksRepository.findAll();
    }

    public Books findBook(Long id){

        return booksRepository.findById(id).orElseThrow(
            () -> new ContentNotFound("Id not found")
        );
    }

    public Books updateBook(Long id, Books books){
        Books bookToUpdate = this.findBook(id);

        if (books.getPublishYear()!= null){
            bookToUpdate.setPublishYear(books.getPublishYear());
        }
        if (books.getAuthor() != null){
            bookToUpdate.setAuthor(books.getAuthor());
        }
        if (books.getGenre() != null){
            bookToUpdate.setGenre(books.getGenre());
        }
        if (books.getTitle()!= null) {
            bookToUpdate.setTitle(books.getTitle());
        }
        return booksRepository.save(bookToUpdate);
    }

    public void deleteBook(Long id){
        var book = this.findBook(id);
        booksRepository.delete(book);
    }

    public List<Books> mostBorroweBooks(){
        var list = this.findAllBooks();

        return list.stream()
                .sorted(Comparator.comparing(Books::getBorrowedTimes).reversed())
                .collect(Collectors.toList());

    }

}
```

Classes dos Serviços

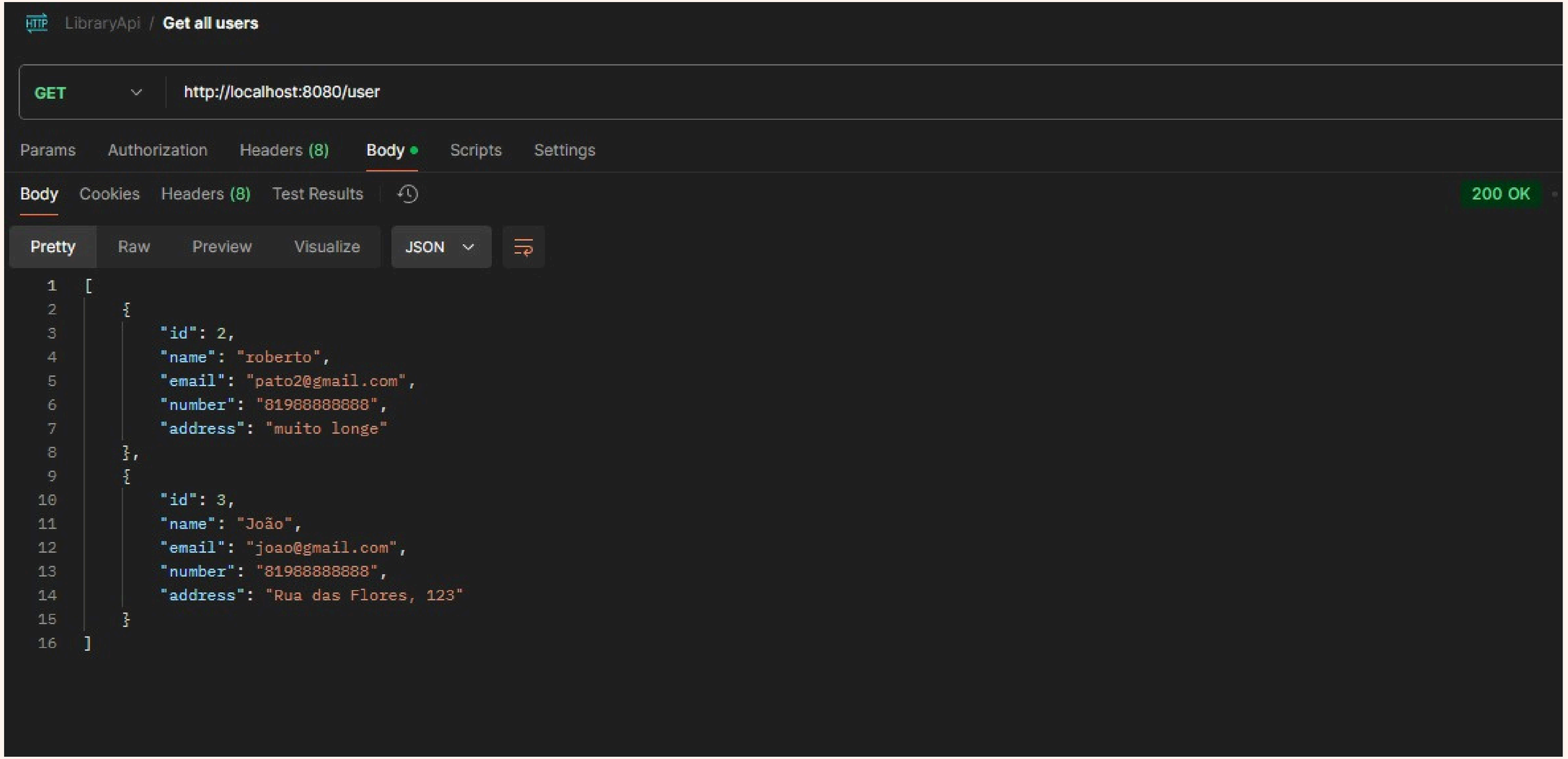
```java
@Service
@RequiredArgsConstructor
public class BorrowedBooksService {

    private final BorrowedBooksRepository borrowedBooksRepository;
    private final BooksService booksService;
    private final UserService userService;

    public List<BorrowedBooks> findAllBorrowedBooks(){
        return borrowedBooksRepository.findAll();
    }


    public BorrowedBooks borrowNewBook(Long user_id, Long book_id, LocalDate returnDate){

        var user = userService.findUser(user_id);
        var book = booksService.findBook(book_id);
        var borrowPerUser = 0;

        for (BorrowedBooks l: findAllBorrowedBooks()){
            if (Objects.equals(l.getUser().getId(), user.getId())){
                borrowPerUser += 1;
            }
            if (borrowPerUser>=2){
                 throw new MaxBorrows("Users already borrow the max limit");
            }
        }

        BorrowedBooks borrowedBooks = new BorrowedBooks();
        borrowedBooks.setBooks(book);
        borrowedBooks.setUser(user);
        borrowedBooks.setReturnDate(returnDate);

        book.setBorrowedTimes(book.getBorrowedTimes()+1);
        return borrowedBooksRepository.save(borrowedBooks);

    }

    public void returnBook(Long user_id, Long book_id){

        for (BorrowedBooks l : this.findAllBorrowedBooks()){
            if (Objects.equals(l.getUser().getId(), user_id) && Objects.equals(l.getBooks().getId(), book_id)){
                borrowedBooksRepository.delete(l);
                return;
            }

        }

         throw new ContentNotFound(
                 "No users found."
        );
    }

}
```

```java
@Service
@RequiredArgsConstructor
public class PendingBooksService {

    private final UserService userService;
    private final BooksService booksService;
    private final PendingBooksRepository pendingBooksRepository;

    public List<PendingBooks> findAllPendingBooks(){
        return pendingBooksRepository.findAll();
    }

    public PendingBooks newPendingBook(Long userId, Long bookId){

        var user = userService.findUser(userId);
        var book = booksService.findBook(bookId);

        for (PendingBooks e: this.findAllPendingBooks()){
            if (e.getUser()==user){
                 throw new MaxBorrows("Limite maximo de emprestimos pendentes foi atingido");
             }
        }

        PendingBooks pendingBooks = new PendingBooks();
        pendingBooks.setUser(user);
        pendingBooks.setBooks(book);

        return pendingBooksRepository.save(pendingBooks);
    }

        public void returnPendingBook(Long user_id, Long book_id){

        for (PendingBooks l : this.findAllPendingBooks()){
            if (Objects.equals(l.getUser().getId(), user_id) && Objects.equals(l.getBooks().getId(), book_id)){
                pendingBooksRepository.delete(l);
                return;
            }

        }

         throw new ContentNotFound(
                 "No users found."
        );
    }
}
```

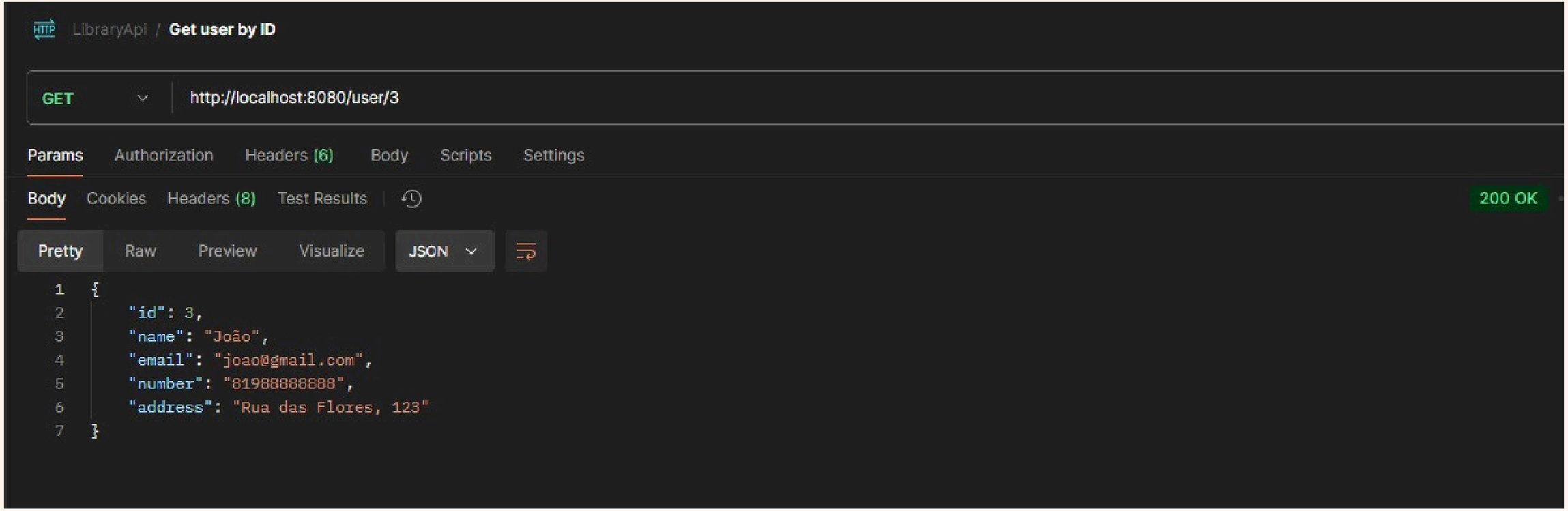
Demonstração de uso



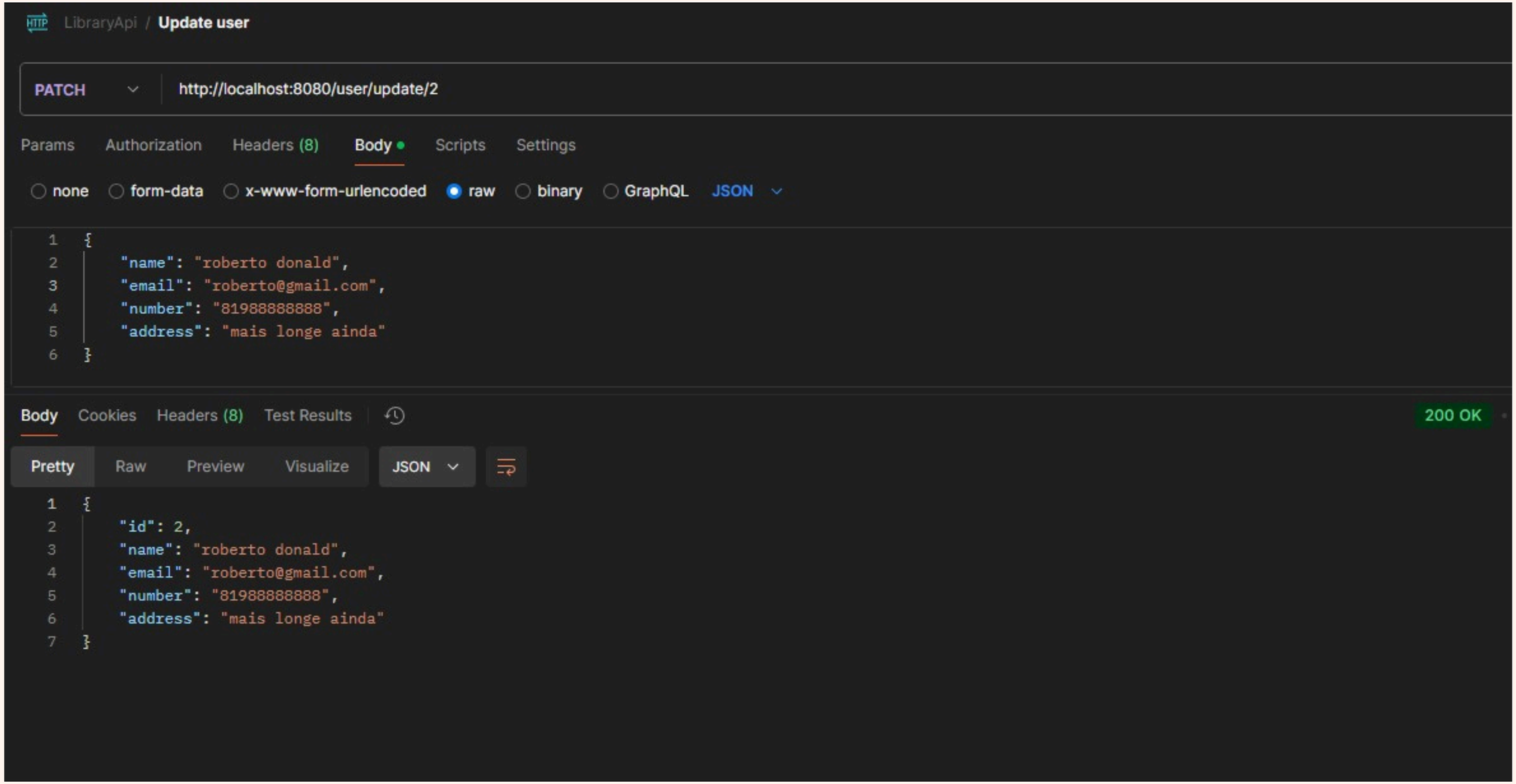
Demonstração de uso



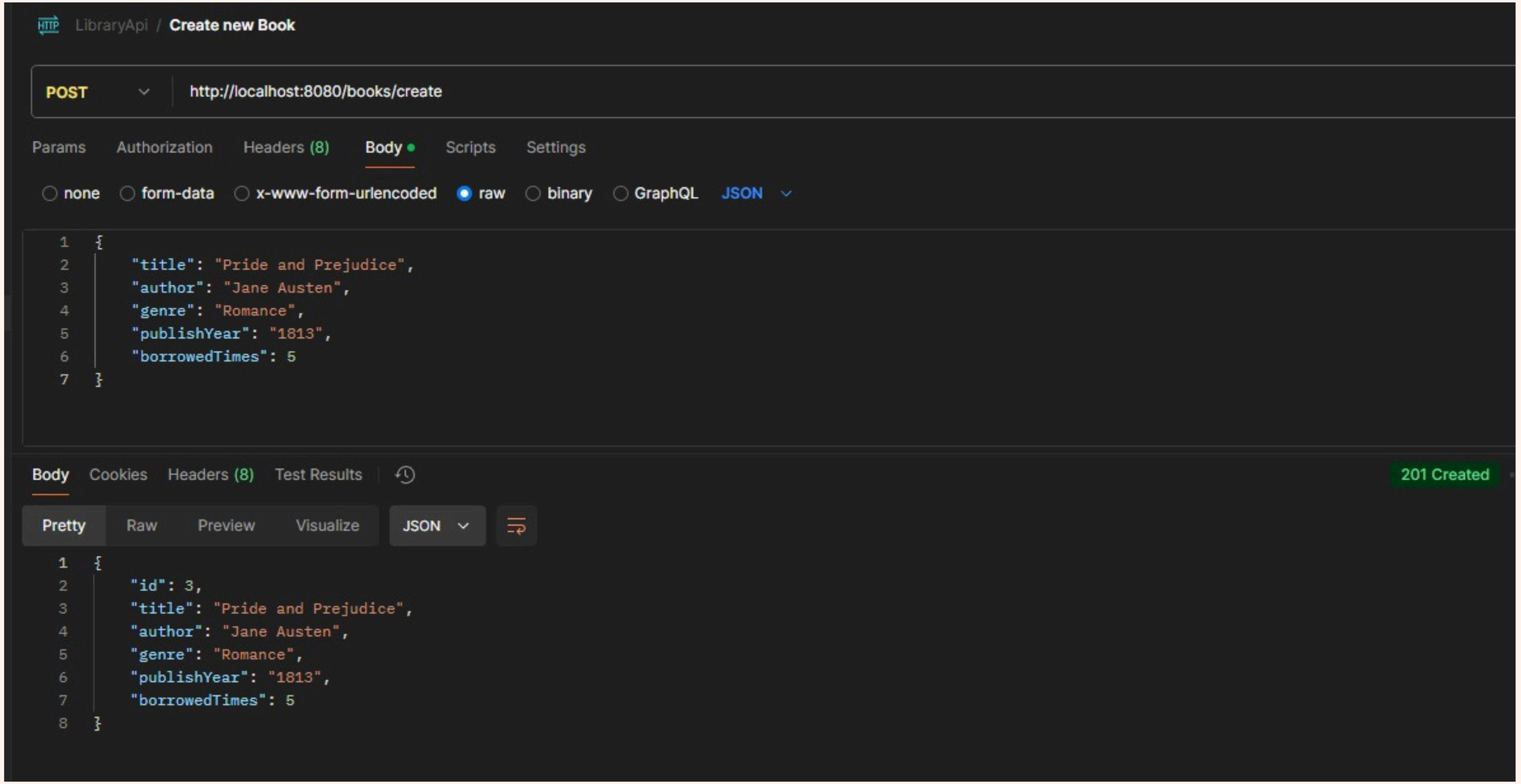
Demonstração de uso



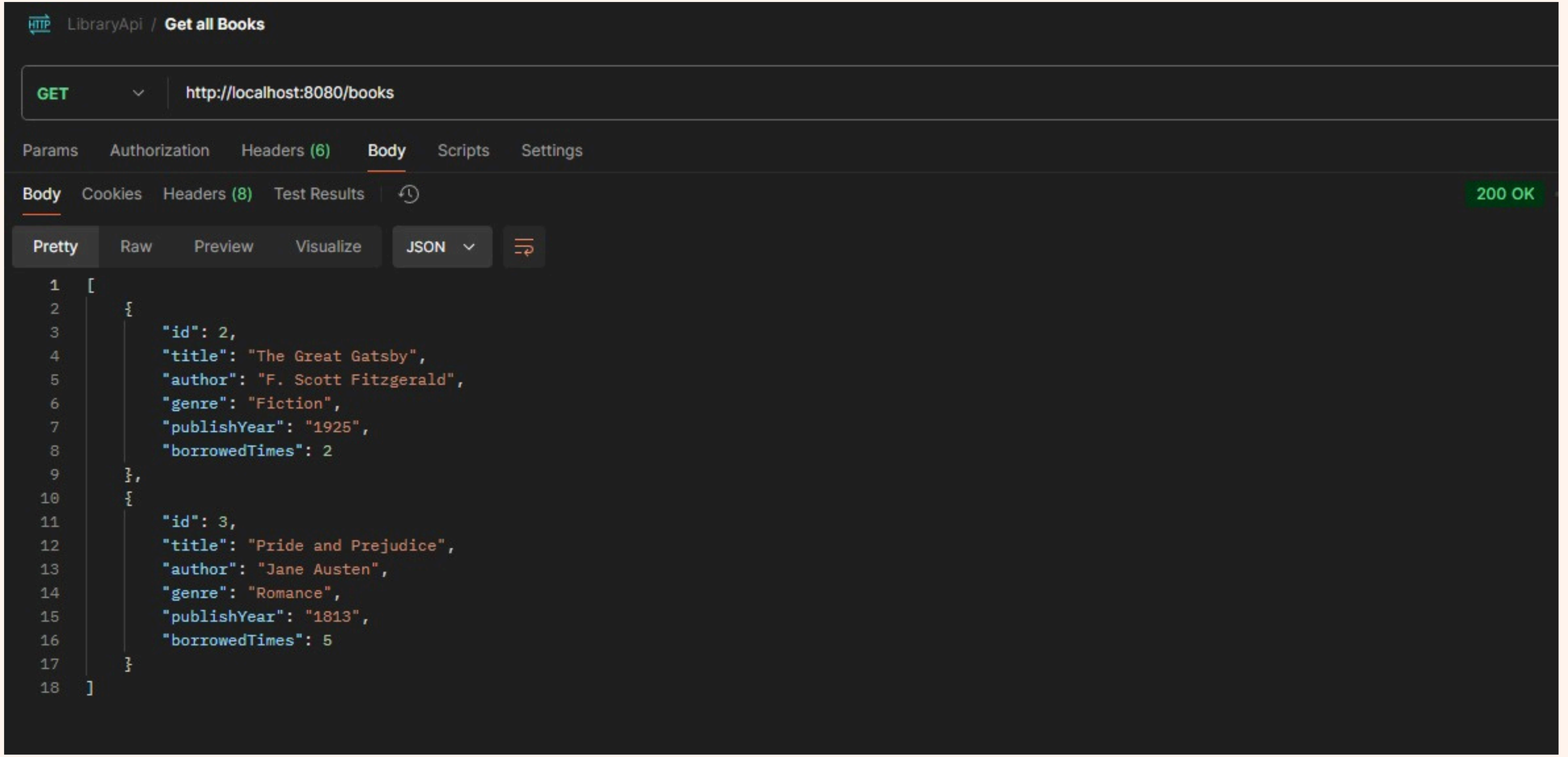
Demonstração de uso



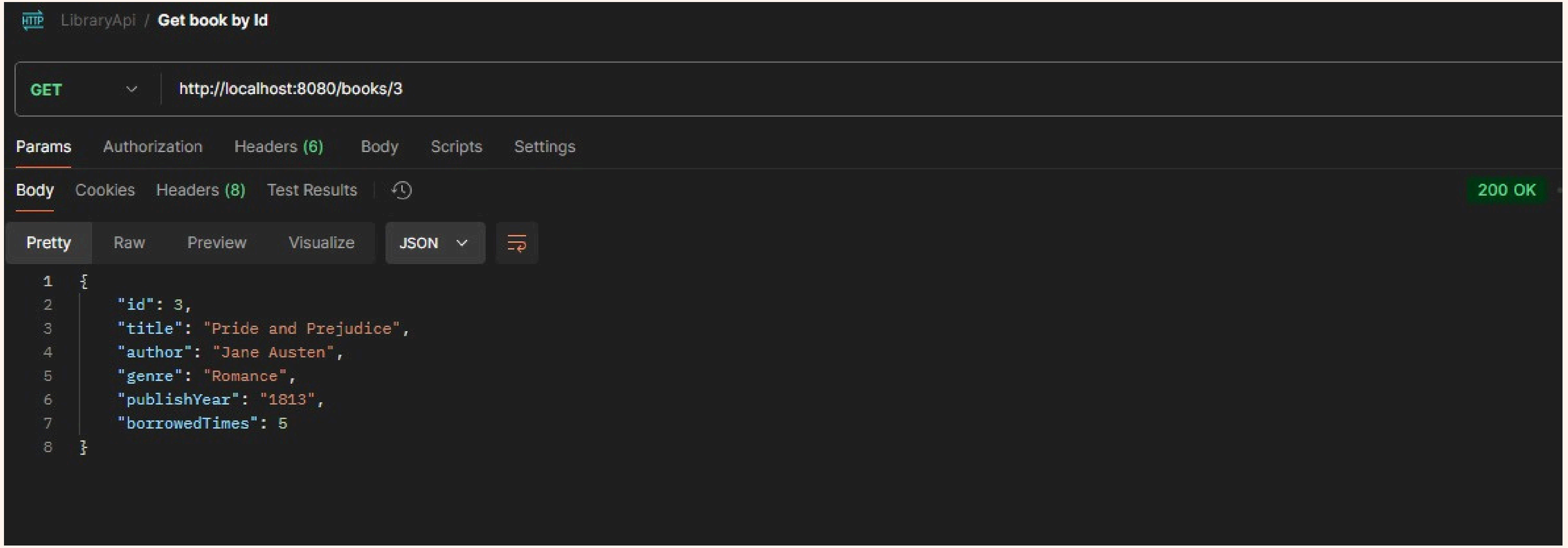
Demonstração de uso



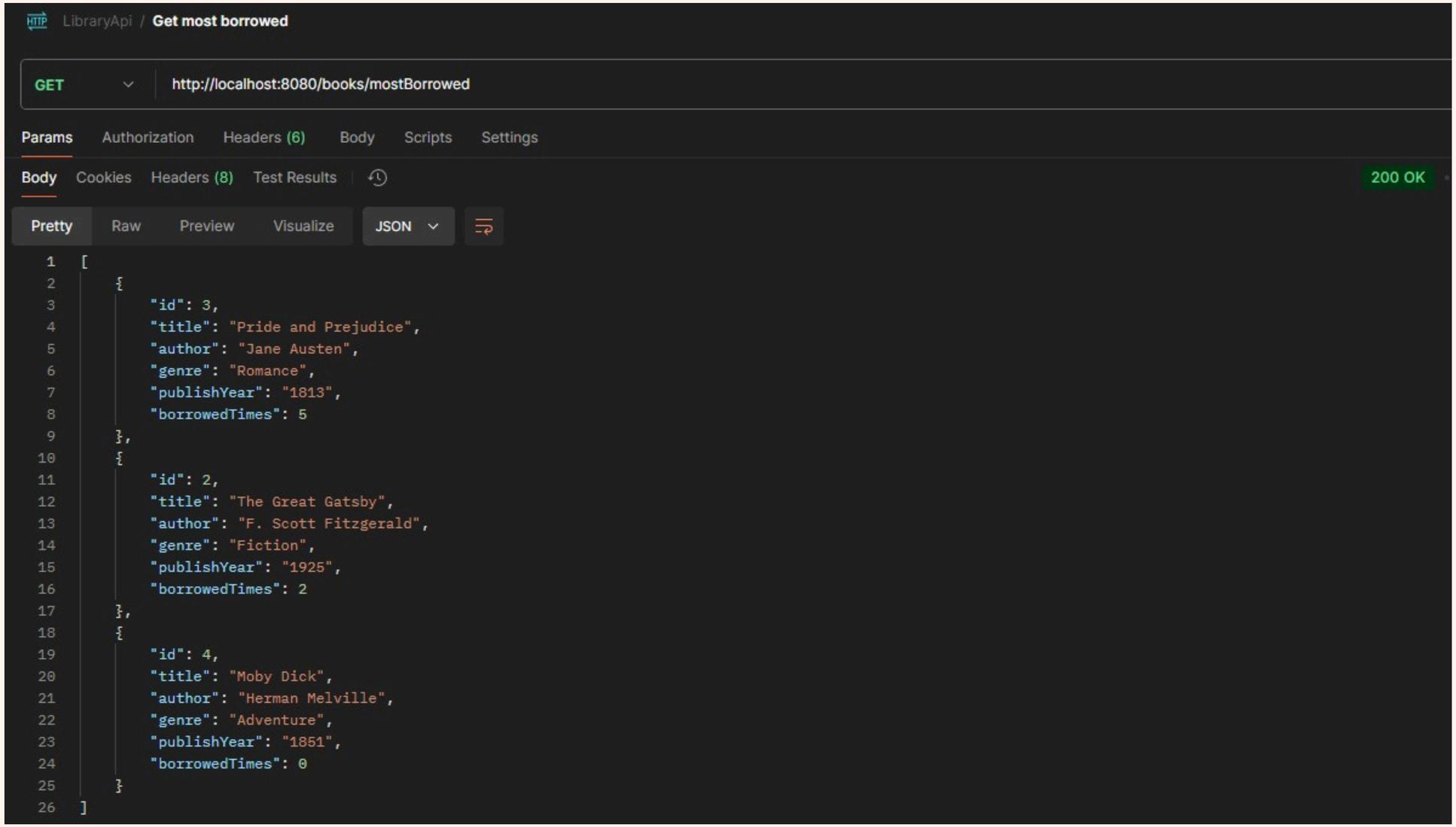
Demonstração de uso



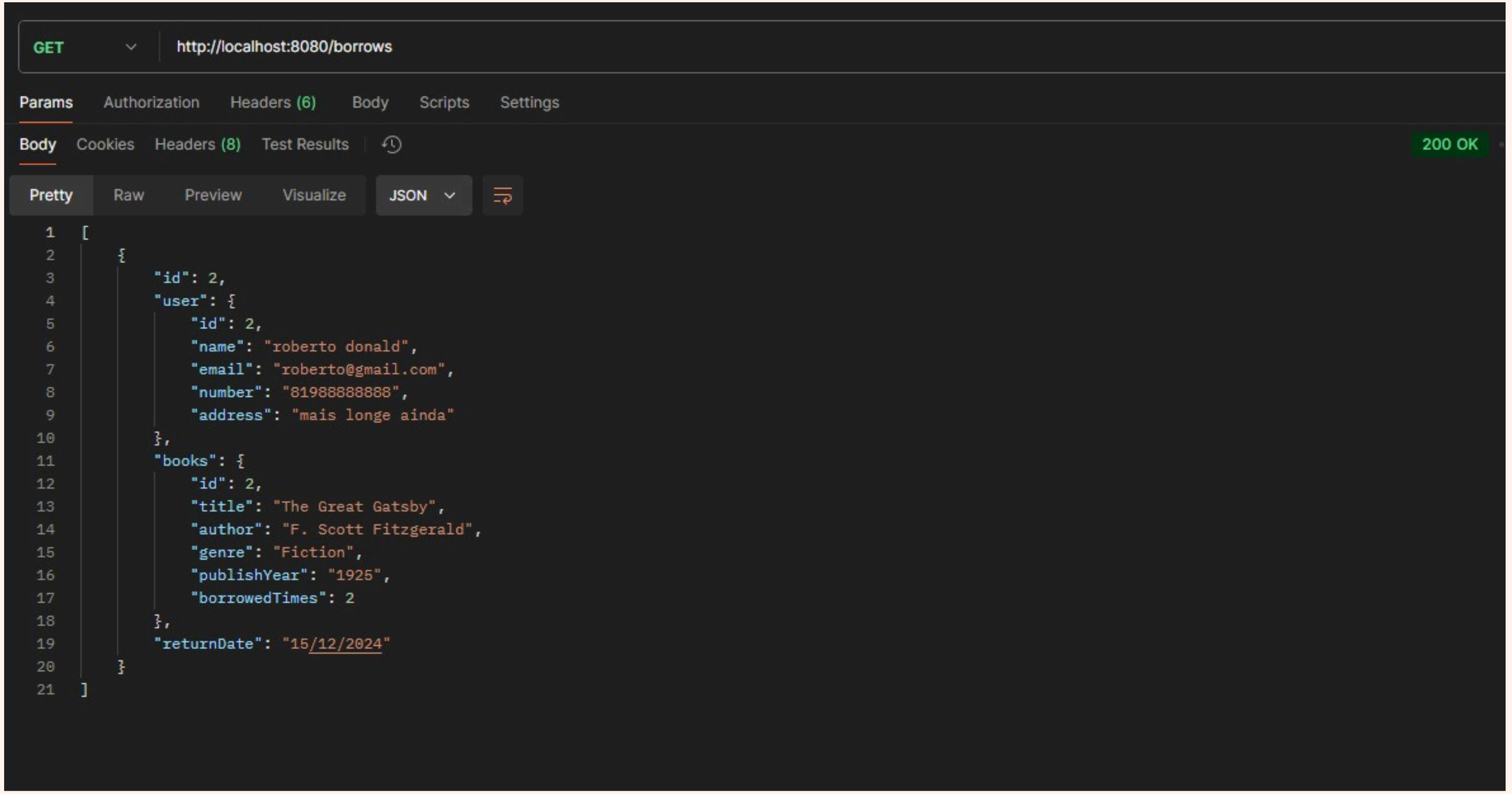
Demonstração de uso



Demonstração de uso



Demonstração de uso



Lições Aprendidas

Aprendemos a importância de dividir o trabalho em grupo, definir prioridades e separar atividades para que cada membro tenha um foco. Além disso, a organização do trabalho com o Trello utilizando o método Kanban ajuda muito na organização e do alinhamento da equipe com as atividades de cada um.

Referências

- YouTube
- GitHub
- ChatGPT
- Figma - EAP
- Documentação Spring Boot
- Monday - Gráfico Gantt

Obrigado!