```
// Main Application
@SpringBootApplication
public class LibraryManagementApplication {
    public static void main(String[] args) {
        SpringApplication.run(LibraryManagementApplication.class, args);
    }
}

// Book Entity
@Entity
public class Book {
    @Id @GeneratedValue
    private Long bookId;
    private String title;
    private String author;
    private String category;
    private boolean availability;
}

// User Entity
@Entity
public class User {
    @Id @GeneratedValue
    private Long userId;
    private String name;
    private String membershipType;
}
```

```
// Transaction Entity
@Entity
public class Transaction {
    @Id @GeneratedValue
    private Long transactionId;
    @ManyToOne
    private Book book;
    @ManyToOne
    private User user;
    private LocalDate issueDate;
    private LocalDate returnDate;
    private String status;
}

// Repositories
public interface BookRepository extends JpaRepository<Book, Long> {
    List<Book> findByTitleContaining(String title);
    List<Book> findByAuthorContaining(String author);
    List<Book> findByCategory(String category);
}

public interface UserRepository extends JpaRepository<User, Long> {}

public interface TransactionRepository extends JpaRepository<Transaction, Long>
{
    List<Transaction> findByReturnDateBefore(LocalDate date);
    List<Transaction> findByUser(User user);
}
```

```
// Book Controller
@RestController
@RequestMapping("/books")
public class BookController {
    @Autowired private BookRepository bookRepo;

    @PostMapping
    public Book addBook(@RequestBody Book book) { return bookRepo.save(book); }

    @GetMapping
    public List<Book> getAllBooks() { return bookRepo.findAll(); }

    @GetMapping("/search")
    public List<Book> search(@RequestParam String title) {
        return bookRepo.findByTitleContaining(title);
    }

    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable Long id) { bookRepo.deleteById(id); }
}

// User Controller
@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired private UserRepository userRepo;

    @PostMapping
```

```
    public User addUser(@RequestBody User user) { return userRepo.save(user); }

    @GetMapping
    public List<User> getAllUsers() { return userRepo.findAll(); }

    @DeleteMapping("/{id}")
    public void deleteUser(@PathVariable Long id) { userRepo.deleteById(id); }
}

// Transaction Controller
@RestController
@RequestMapping("/transactions")
public class TransactionController {
    @Autowired private TransactionRepository transRepo;
    @Autowired private BookRepository bookRepo;
    @Autowired private UserRepository userRepo;

    @PostMapping("/lend")
    public Transaction lend(@RequestParam Long bookId, @RequestParam Long
userId) {
        Book book = bookRepo.findById(bookId).get();
        User user = userRepo.findById(userId).get();
        if (!book.isAvailability()) throw new RuntimeException("Book not available");
        book.setAvailability(false);
        bookRepo.save(book);
        Transaction tx = new Transaction();
        tx.setBook(book);
        tx.setUser(user);
        tx.setIssueDate(LocalDate.now());
```

```
        tx.setStatus("Issued");
        return transRepo.save(tx);
    }

    @PostMapping("/return")
    public Transaction returnBook(@RequestParam Long txId) {
        Transaction tx = transRepo.findById(txId).get();
        tx.setReturnDate(LocalDate.now());
        tx.setStatus("Returned");
        Book book = tx.getBook();
        book.setAvailability(true);
        bookRepo.save(book);
        return transRepo.save(tx);
    }

    @GetMapping("/overdue")
    public List<Transaction> overdue() {
        return transRepo.findByReturnDateBefore(LocalDate.now());
    }

    @GetMapping("/user/{id}")
    public List<Transaction> userHistory(@PathVariable Long id) {
        User user = userRepo.findById(id).get();
        return transRepo.findByUser(user);
    }
}

// Security Config
```

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests().anyRequest().authenticated()
            .and().httpBasic();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
            .withUser("admin").password("admin").roles("LIBRARIAN")
            .and().passwordEncoder(NoOpPasswordEncoder.getInstance());
    }
}
```

Explanation:

This project is a simple Spring Boot application that helps manage a library. You can add books and users, lend books to users, return them, and keep track of what's available. The system uses an H2 database to store everything and Spring Security to make sure only authorized people can manage the library. You can also search for books by title or author and see reports like who borrowed what and what's overdue.