**----- class LentBookView-----**

/\*\*\* Submit new borrow registration information to controller via POST request  
\* @param password a string representation of user’s password.  
\* @param cardID a long value represent ID number of borrower card  
\* @param bookIDs the array of book ids the user wants to borrow   
\*/  
public static void postBookBorrowRegistration(String $password, long $cardID, $copies)  
{

Submit new borrow registration information to controller via POST request

}

/\*\*  
\* Notifies success/fail to user  
\* @param message a string representation of the message to be notified to the user.  
\*/  
public void notifyUser(String message)  
{

notify message to user;

}

**----- class BookBorrowRegistrationController -----**

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request)

{

$user = Auth::user();

$validator = Validator::make($request->all(), [

'borrow-card' => 'required',

'password-confirmation' => 'required'

]);

if (!Hash::check($request->input('password-confirmation'), $user- >getAuthPassword()))

{

$validator->errors()->add('password', 'The password confirmation not match!');

}

$borrowCard = BorrowCard::where('card\_number', $request→input('borrow- card'))->first();

if (is\_null($borrowCard)){

$validator->errors()->add('borrow-card', 'The borrow card number is not valid!');

}else {

if(!$borrowCard->validateBorrowCard($user->getAuthIdentifier())) {

$validator->errors()->add('borrow-card', 'This borrow card is not belong to you!');

}

if(!$borrowCard->checkExpirationDate()) {

$validator->errors()->add('borrow-card', 'Your borrow card is expired!');

}

}

$bookIDs = json\_decode($request->input('book-ids'));

$numberOfUserBorrowedCopies = (new BorrowRecord())- >getNumberOfUserBorrowedCopies($user->getAttribute('id'));

if ($numberOfUserBorrowedCopies + count($bookIDs) > 5) {

$validator->errors()->add('borrow-copies', 'Borrow exceed maximum number of copies allowed');

}

$bookObject = new Book();

DB::beginTransaction();

foreach ($bookIDs as $bookID) {

$book = $bookObject->find($bookID);

if (is\_null($book)) {

$validator->errors()->add('book-ids', 'The book id '.$bookID.' not found!');

DB::rollback();

break;

}

$bookCopy = $book->getAnAvailableCopy();

if (is\_null($bookCopy)) {

$validator->errors()->add('book-ids', 'The book id '.$bookID.' not have any available copy!');

DB::rollback();

break;

}

$bookCopy->updateBookCopyStatus('borrowed');

$borrowRecord = new BorrowRecord();

$borrowRecord->createNewBorrowRecord($bookCopy->getAttribute('id'), $user->getAttribute('id'), 'registered');

}

if ($validator->errors()->count()) {

return redirect()->back()->with('errors', $validator->errors());

}

DB::commit();

return redirect()->route('book.borrow.register.create')->with('success', "Register successfully!");

}

**----- Class User -----**

/\*\*  
\* Get password of an user  
\* @return the password of an user.  
\*/  
public String getPassword(){

return this.password;

}

/\*\*\* Get card ID of an user  
\* @return the card ID of an user  
\*/  
public long getCardID(){

return this.cardID;

}

**----- Class BookCopy -----**

/\*\*\* update the status of a book copy (available, referenced, borrowed or lent) in the database  
\*/  
public function updateBookCopyStatus(string $status)

{

$this->setAttribute('copy\_status', $status);

$this->save();

}

**----- Class BorrowCard -----**

**/\*\***

\* Retrieve the borrow card that have the card number in the database

\* @param cardNumber card number of borrow card

\* @return BorrowCard borrow card having card number

\*/

public function getBorrowCardByCardNumber(string $cardNumber){

return $this->BorrowCard::where('card\_number', $cardNumber)->first();

}

/\*\*

\* Check the borrowing card belongs to the current user

\* @return bool true if the user with userID own this borrow card, otherwise false

\*/

public function validateBorrowCard(int $userID){

return $this->user->id == $userID;

}

/\*\*

\* Check if the borrow card is expired

\* @return bool true if the borrow card is not expired (card is valid), otherwise false

\*/

public function checkExpirationDate(){

return new DateTime($this->getAttribute('expired\_date')) > new DateTime();

}

**----- Class BorrowRecord -----**

/\*\*

\* Get number of copies that an user borrowed

\* @param userID id of user

\* @return int number of copies that an user borrowed

\*/

public function getNumberOfUserBorrowedCopies(int $userID){

return $this->where('user\_id', $userID)->count();

}

/\*\*

\* Insert a new book record into the database

\* @param bookCopyID id of a book copy

\* @param userID id of an user

\* @param status status of a borrow record

\*/

public function createNewBorrowRecord(int $bookCopyID, int $userID, string $status){

$this->setAttribute('book\_copy\_id', $bookCopyID)

->setAttribute('user\_id', $userID)

->setAttribute('status', $status);

$this.save();

}

**----- Class Book -----**

**/\*\***

\* Get an available copy to borrow

\* @return BookCopy

\*/

public function getAnAvailableCopy()

{

return $this->copies()

->where('copy\_status', 'available')

->first();

}