SISTEM PEMESANAN TUTOR PRIVATE

TutorLink

oleh:
M. Ihsan Rizqullah Adfa 2208107010029
Muhammad Ilzam 2208107010087

DESKRIPSI SISTEM

TutorLink adalah platform yang menghubungkan siswa dan tutor privat secara terpadu. Siswa dapat mencari tutor berdasarkan mata pelajaran, ketersediaan, dan rating, lalu memesan sesi belajar langsung. Tutor mengelola profil, jadwal, dan pembayaran dengan mudah. Seluruh proses–mulai pendaftaran, pencarian, pemesanan, manajemen jadwal, komunikasi, pembayaran, hingga penilaian–dijalankan dalam satu ekosistem untuk meningkatkan akses pendidikan personal dan efisiensi pengajaran privat.

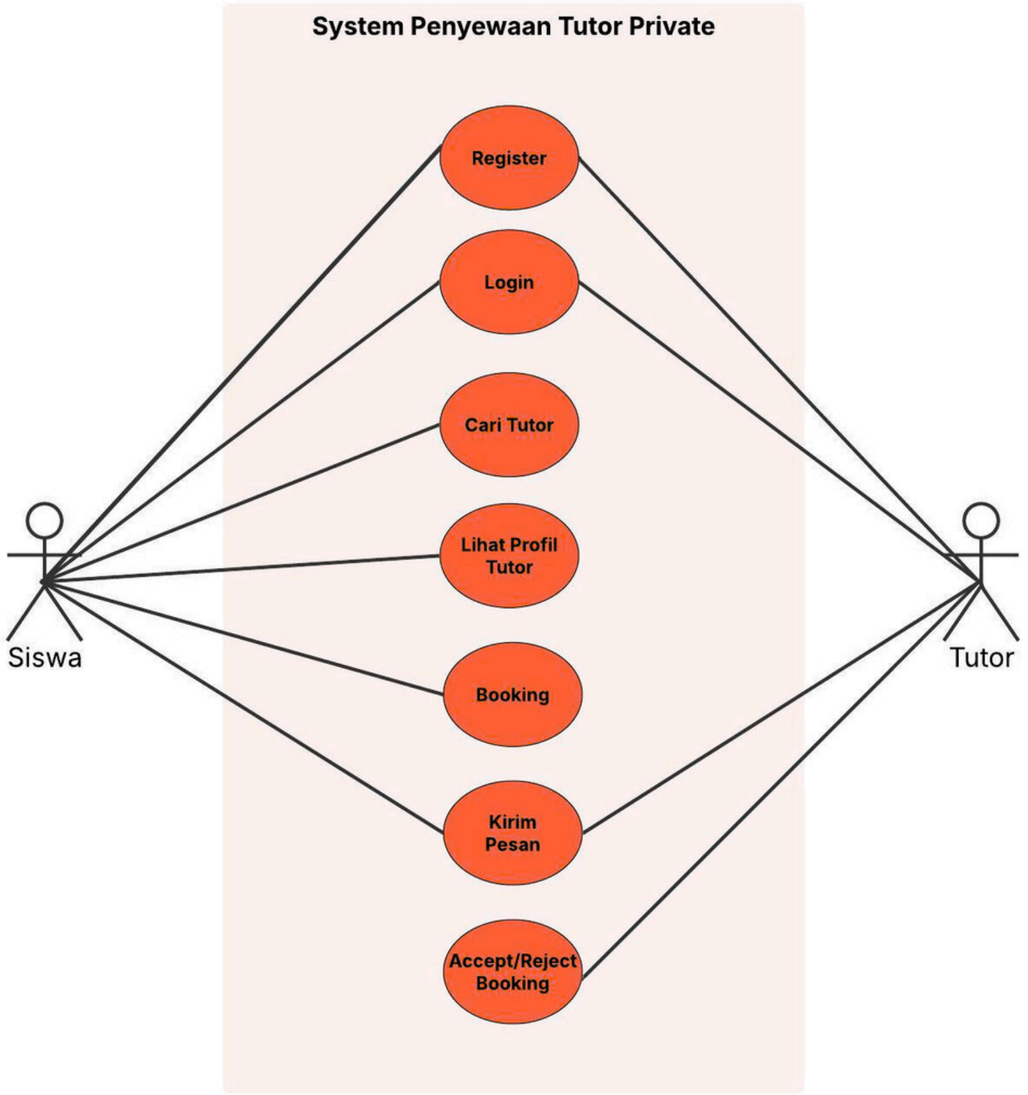
WORKFLOW

1. **Component Identification**
2. **Component Interaction**
3. **Component Specification**

COMPONENT IDENTIFICATION

Tahapan ini bertujuan untuk mengidentifikasi dan menentukan komponen-komponen utama dalam sistem berdasarkan fungsionalitas yang dibutuhkan.

USE CASE DIAGRAM



USE CASE DESCRIPTION

1. Booking Tutor

Nama : **Booking Tutor**

Initiator : **Siswa**

Goal : **Memesan sesi les privat dengan Tutor**

Main Success Scenario

1. Siswa memilih menu “Cari Tutor”.
2. Siswa melihat daftar tutor dan memilih salah satu.
3. Sistem menampilkan detail profil dan jadwal tutor.
4. Siswa memilih tanggal dan waktu yang diinginkan.
5. Sistem mengonfirmasi pemesanan.
6. Siswa melakukan pembayaran.
7. Sistem mencatat pemesanan dan menyimpan detail booking.
8. Sistem mengirim notifikasi ke tutor terkait pemesanan baru.
9. Tutor menerima notifikasi dan mengecek detail booking.
10. Sistem menampilkan status booking kepada siswa (pending).

Extensions

4. a. Tutor tidak tersedia di waktu yang dipilih dan sistem akan menampilkan alternatif waktu.
   b. Siswa memilih waktu baru atau membatalkan.
6. a. Pembayaran gagal.
   b. Sistem menampilkan pesan error dan meminta siswa mengulangi pembayaran.
9. a. Jika tutor menolak pemesanan, sistem mengubah status booking menjadi "Ditolak" dan menginformasikan siswa.
   b. Jika pembayaran sudah dilakukan, sistem memproses refund.

2. Register

Nama : **Register**

Initiator : **Siswa / Tutor**

Goal : **Mendaftar akun baru ke sistem**

Main Success Scenario

1. Pengguna memilih menu “Register”.
2. Pengguna memilih peran (Siswa atau Tutor).
3. Pengguna mengisi data.
4. Sistem melakukan validasi data.
5. Sistem menyimpan data pengguna ke database.
6. Sistem menampilkan notifikasi bahwa akun berhasil dibuat.

Extensions

3. Jika email sudah terdaftar, sistem menampilkan error dan meminta ganti email.
4. Format data user tidak valid , sistem menandai input yang salah dan meminta perbaikan.

3. Login

Nama : **Login**

Initiator : **Siswa / Tutor**

Goal : **Masuk ke dalam sistem menggunakan akun**

Main Success Scenario

1. Pengguna membuka halaman login.
2. Pengguna memasukkan email dan password.
3. Sistem mencocokkan dengan database.
4. Sistem memberikan akses ke dashboard sesuai peran (Siswa/Tutor).

Extensions

1. Apabila Email atau Password salah. Sistem menampilkan pesan error dan meminta ulang login.

USE CASE DESCRIPTION

4. Cari Tutor

Nama : **Cari Tutor**

Initiator : **Siswa**

Goal : **Mencari tutor berdasarkan kriteria tertentu**

Main Success Scenario

1. Siswa membuka halaman “Cari Tutor”.
2. Siswa memasukkan filter (mata pelajaran, lokasi, rating, dll)
3. Sistem menampilkan daftar tutor yang sesuai.
4. Siswa memilih salah satu tutor untuk melihat detail.

Extensions

3. Tidak ada tutor yang sesuai input dari siswa. Sistem menyarankan pencarian yang lebih umum

5. Lihat Profil Tutor

Nama : **Lihat Profil Tutor**

Initiator : **Siswa**

Goal : **Melihat detail informasi seorang tutor**

Main Success Scenario

1. Siswa memilih tutor dari daftar pencarian.
2. Sistem menampilkan halaman profil tutor: Nama, foto, mata pelajaran, rating, review, jadwal
3. Siswa bisa lanjut untuk booking atau chat.

6. Kirim Pesan

Nama : **Kirim Pesan**

Initiator : **Siswa / Tutor**

Goal : **Berkomunikasi langsung via pesan di sistem**

Main Success Scenario

1. Pengguna mendapatkan notifikasi jika ada pesan baru.
2. Pengguna memilih lawan bicara (yang pernah booking)
3. Pengguna mengetik dan mengirim pesan.
4. Sistem menyimpan dan menampilkan pesan real-time.

USE CASE DESCRIPTION

7. Accept/Reject Booking

Nama : **Accept/Reject Booking**

Initiator : **Tutor**

Goal : **Mengelola permintaan booking dari siswa**

Main Success Scenario

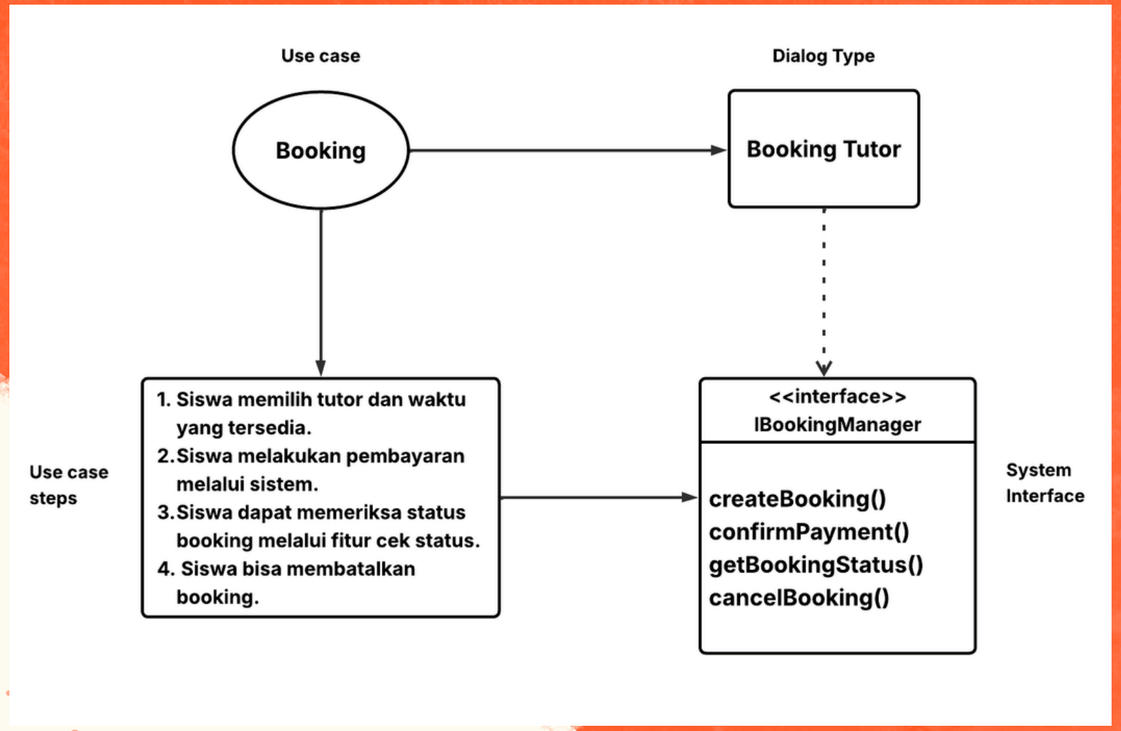
1. Tutor mendapatkan notifikasi booking baru.
2. Tutor membuka detail booking.
3. Tutor memilih tombol “Terima” atau “Tolak”.
4. Sistem menyimpan keputusan dan memberi notifikasi ke siswa.

Extensions

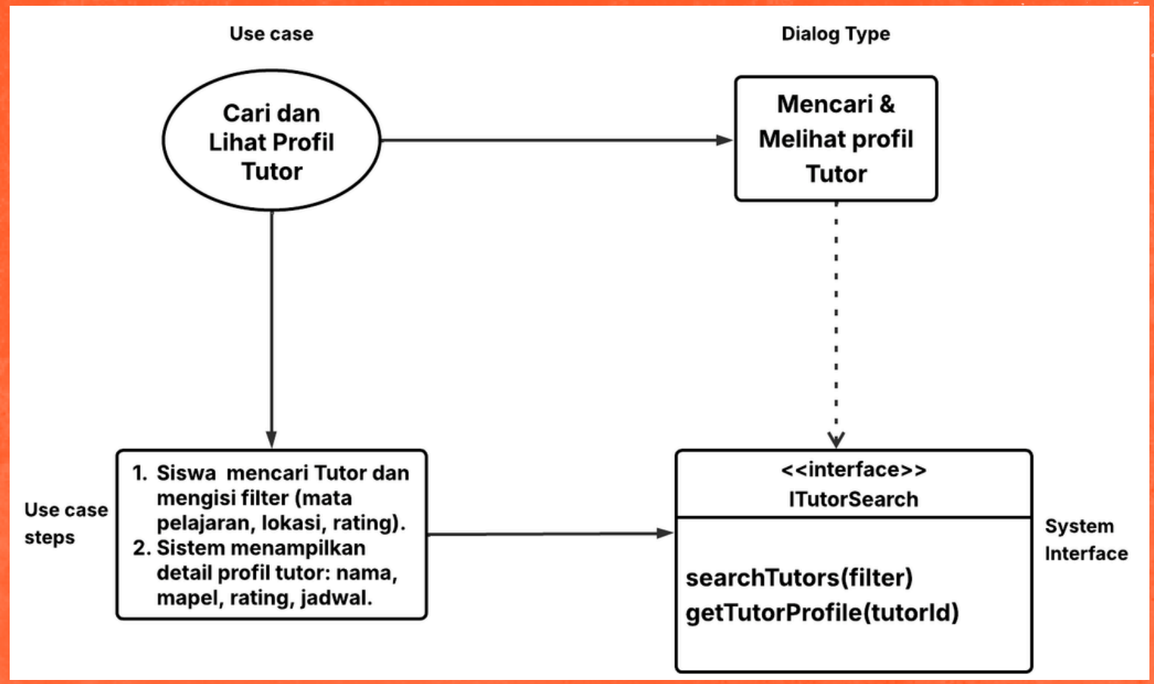
3. Apabila tutor tidak merespon dalam 24 jam. Sistem menandai booking sebagai expired dan memberi tahu siswa.

SYSTEM INTERFACE

1. Booking Tutor

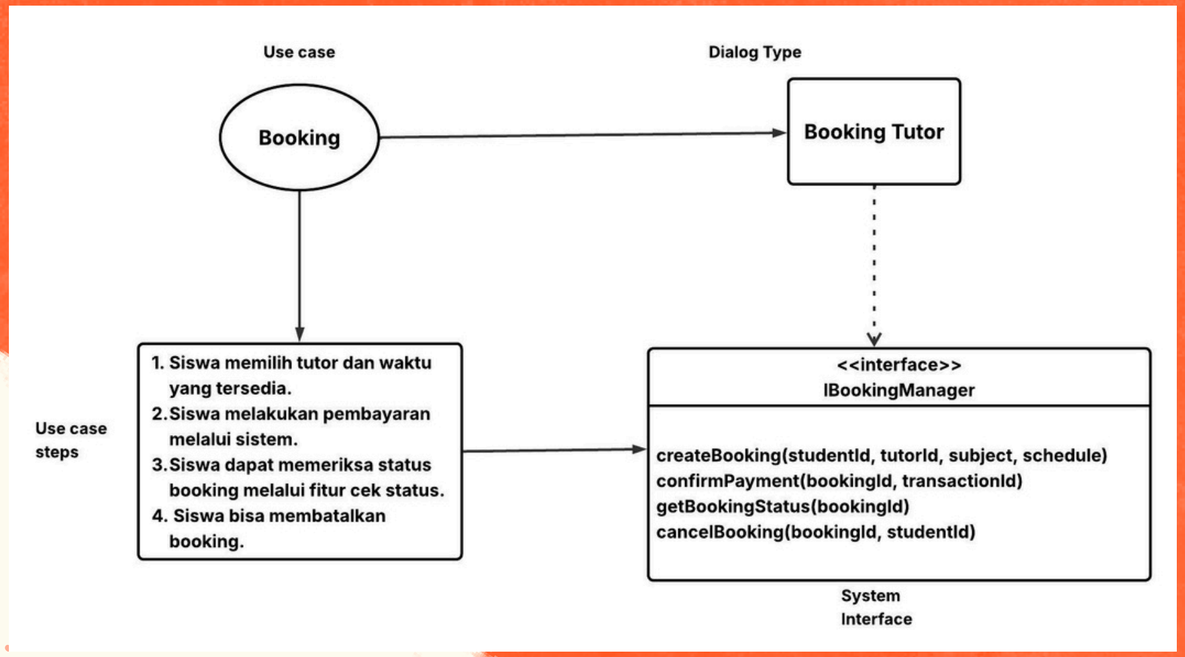


2. Cari Tutor

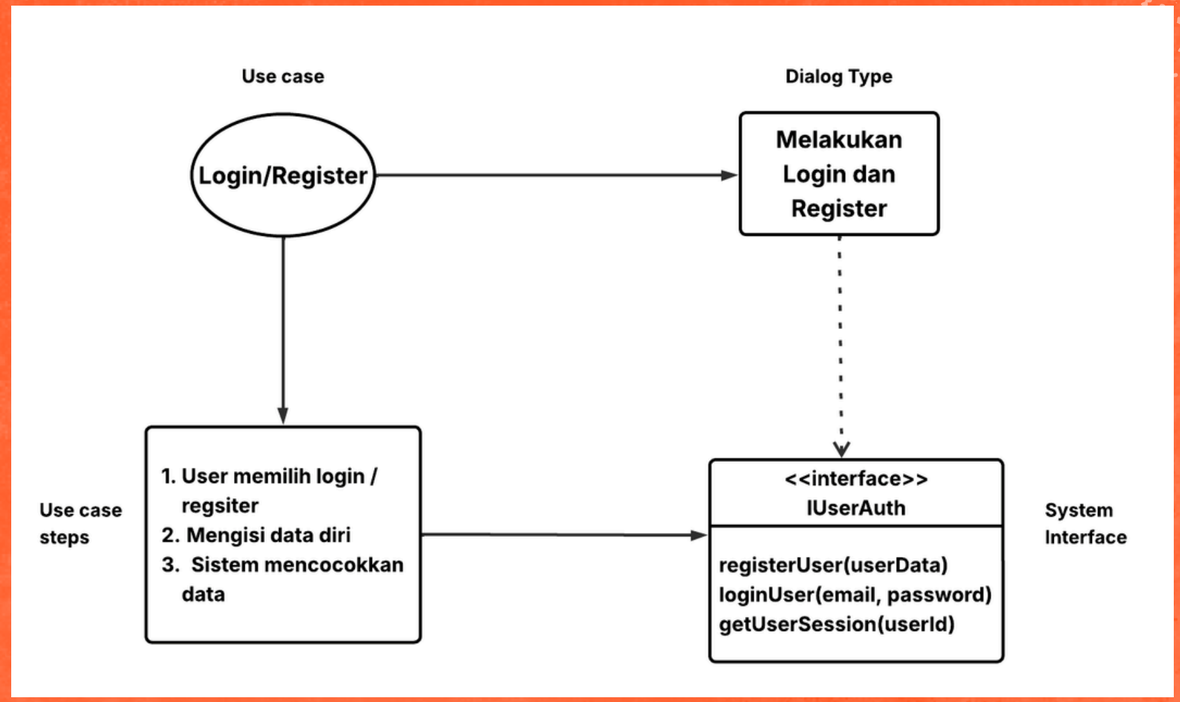


SYSTEM INTERFACE

3. Kirim Pesan

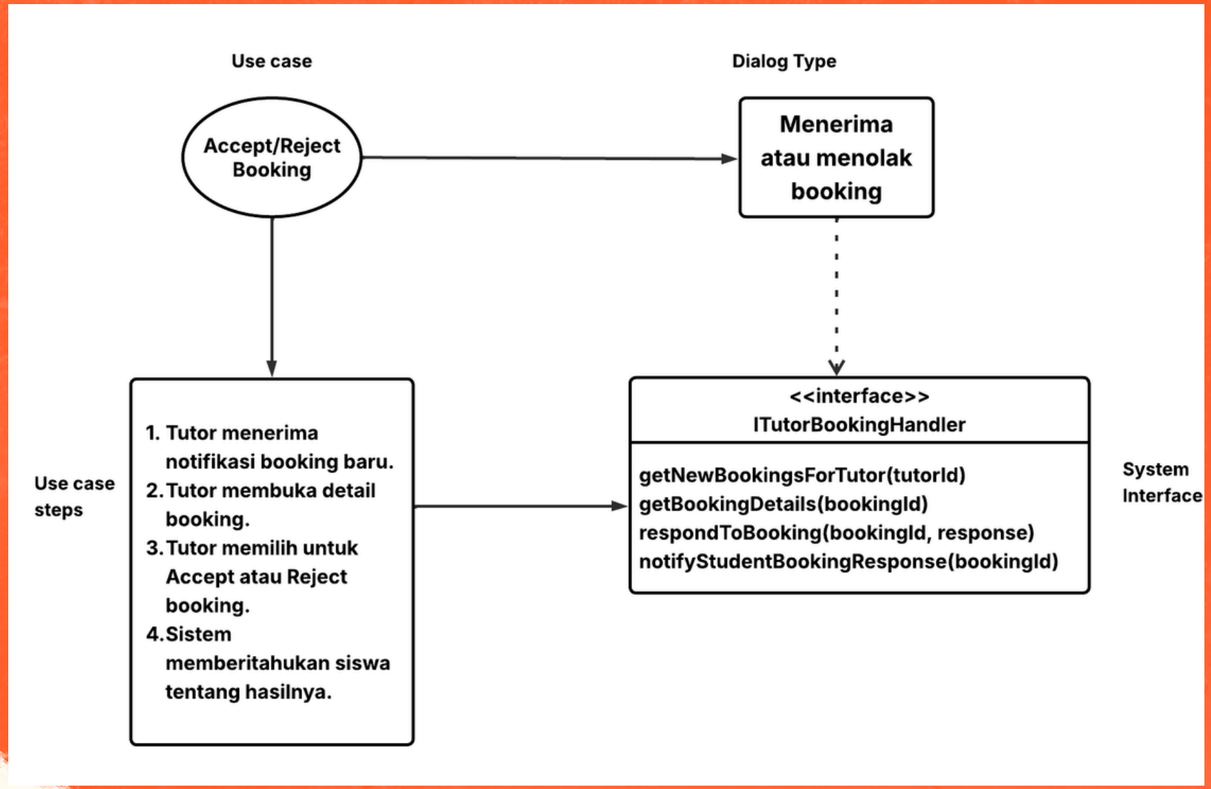


4. Login & Register

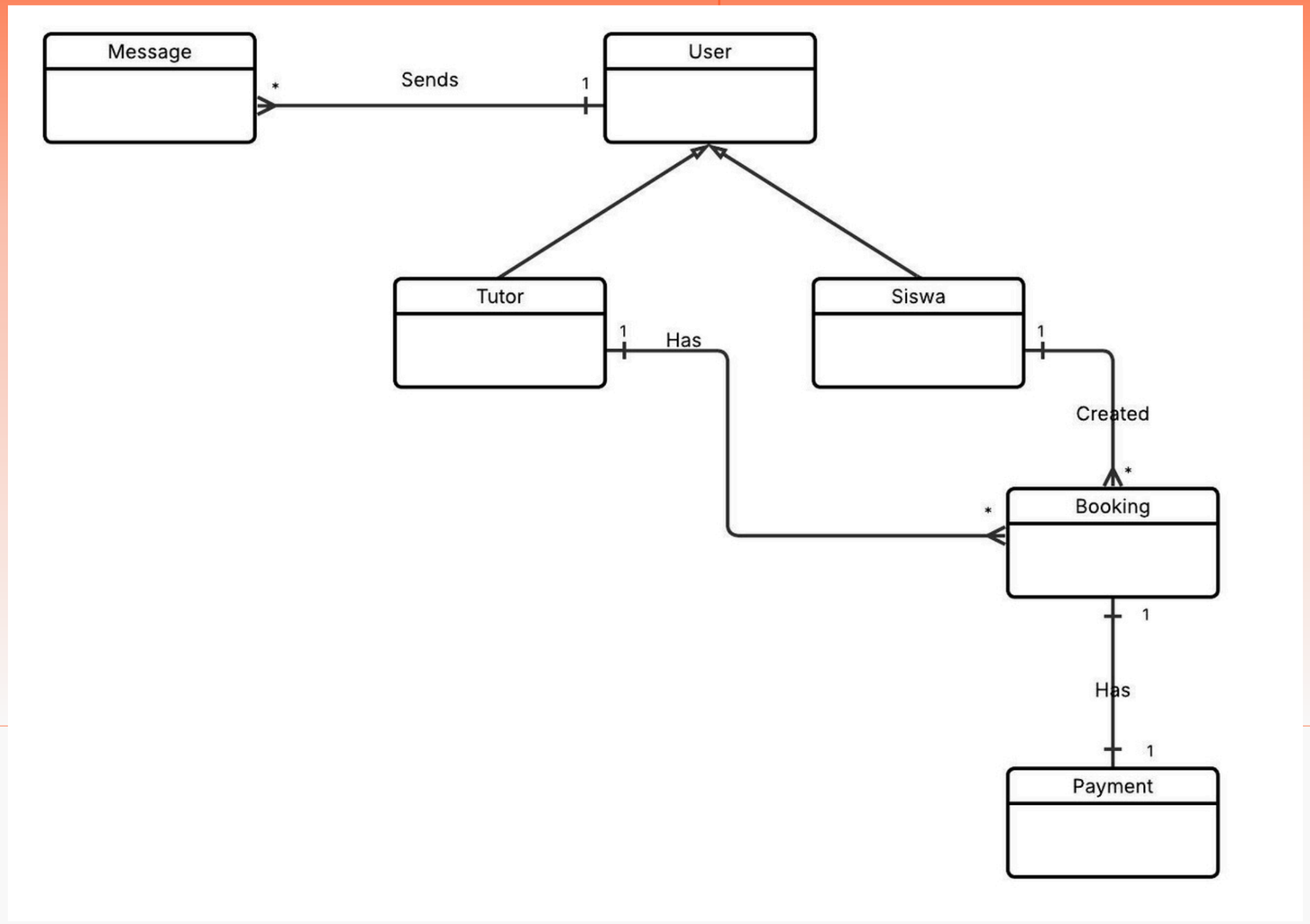


SYSTEM INTERFACE

5. Accept / reject booking

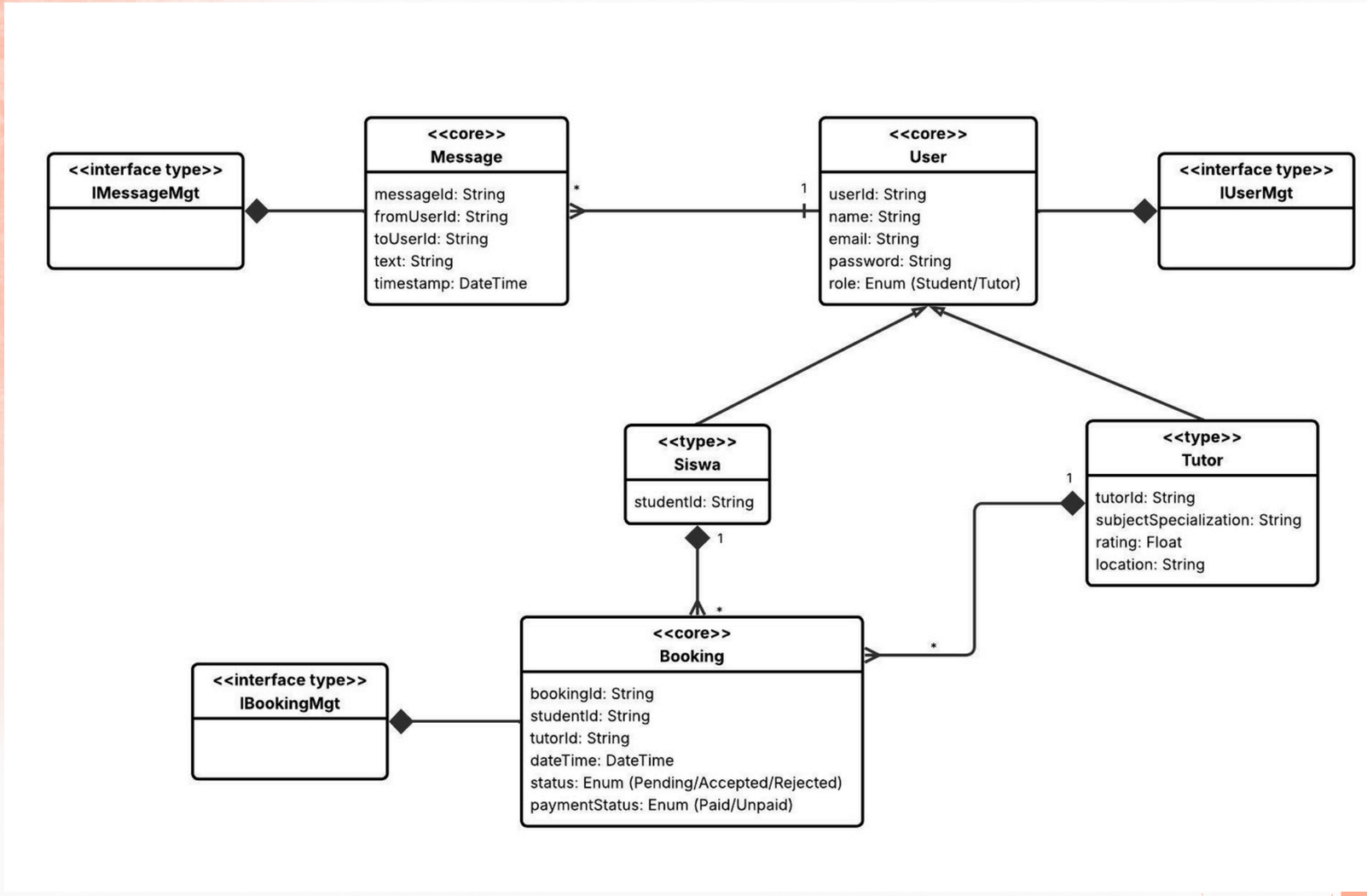


BUSINESS CONCEPT MODEL

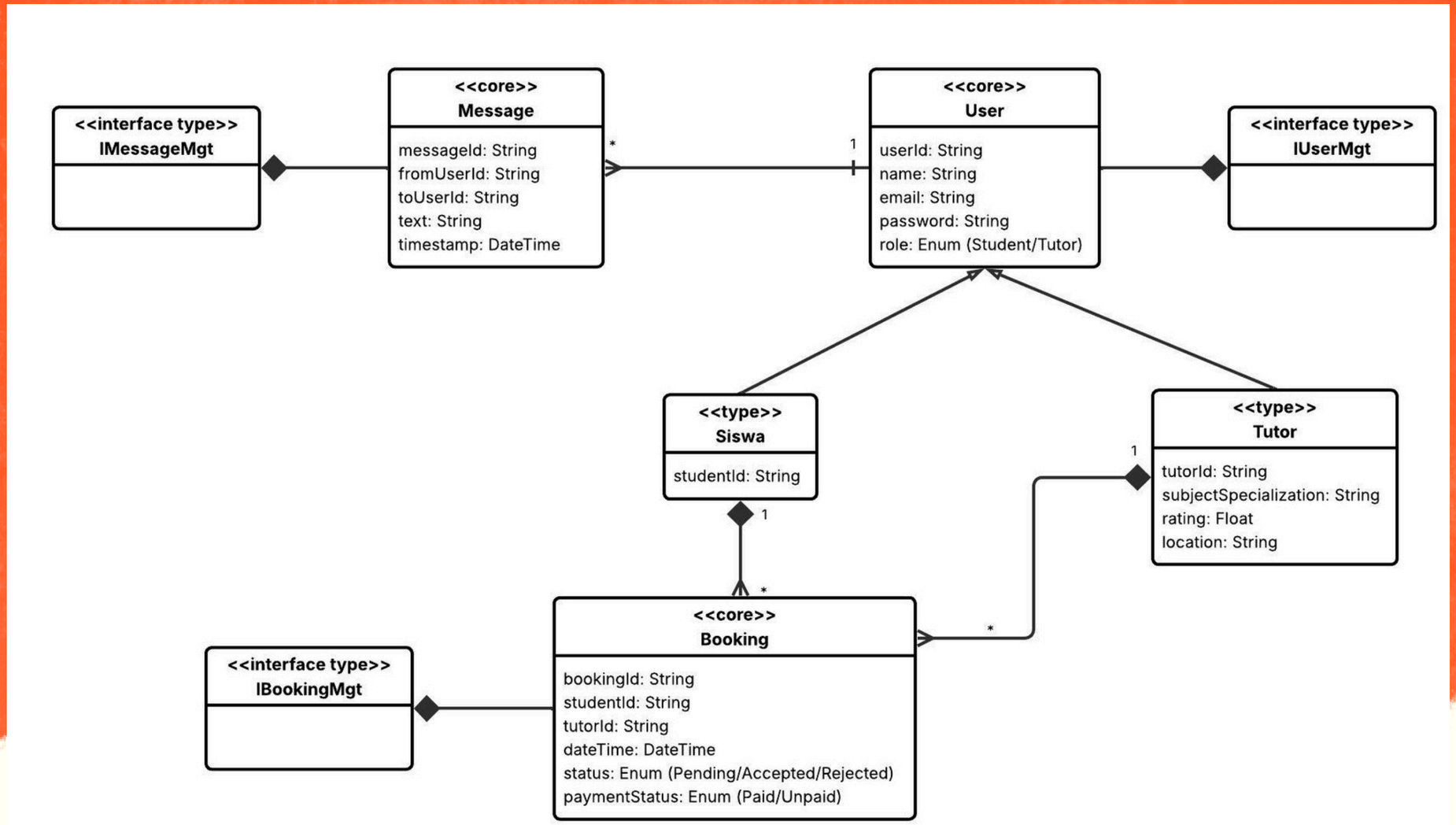


BUSINESS TYPE MODEL

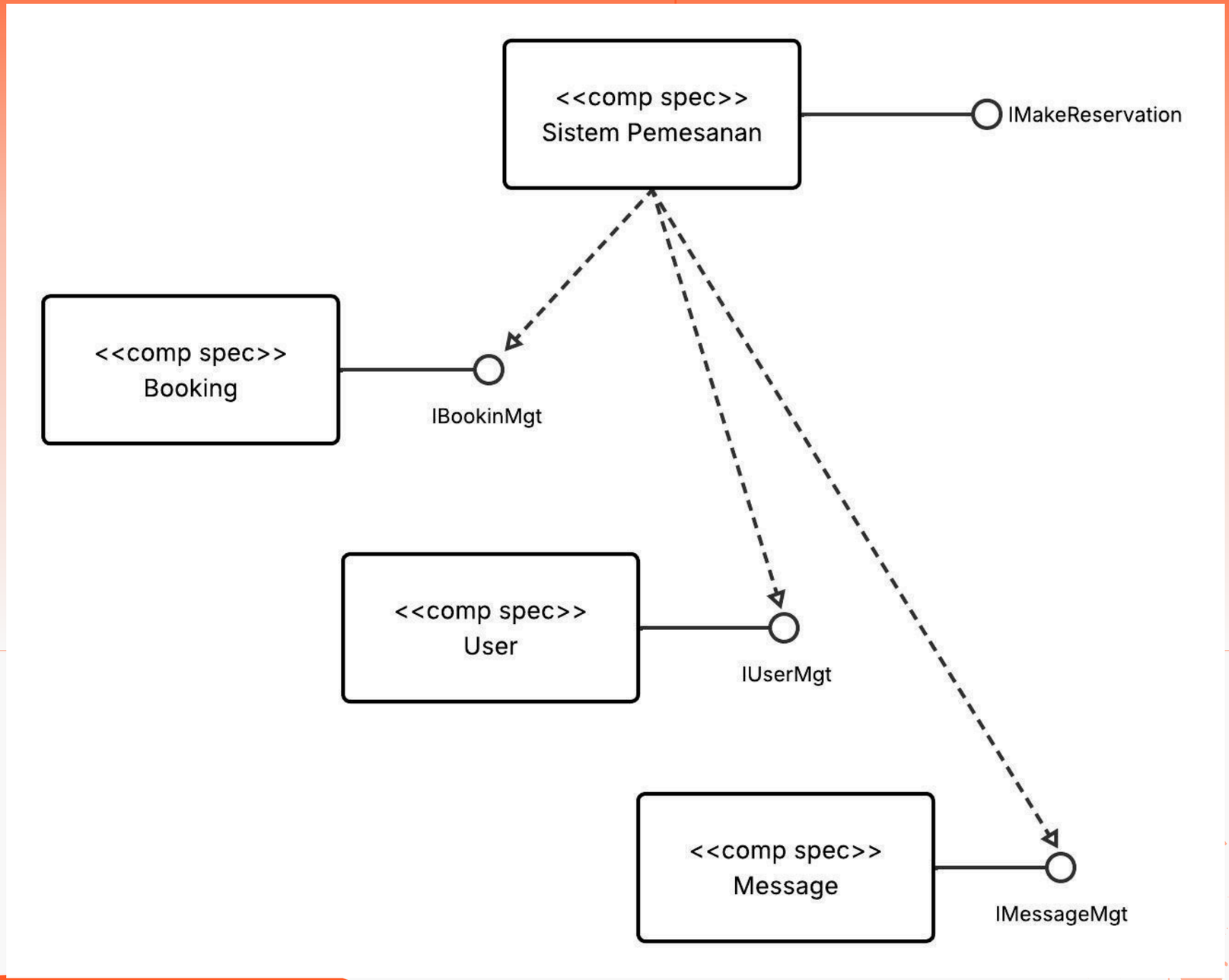
- Mengeleminiasi Payment karena Payment dikelola dalam konteks Booking, jadi tidak perlu entitas terpisah.



BUSINESS INTERFACE



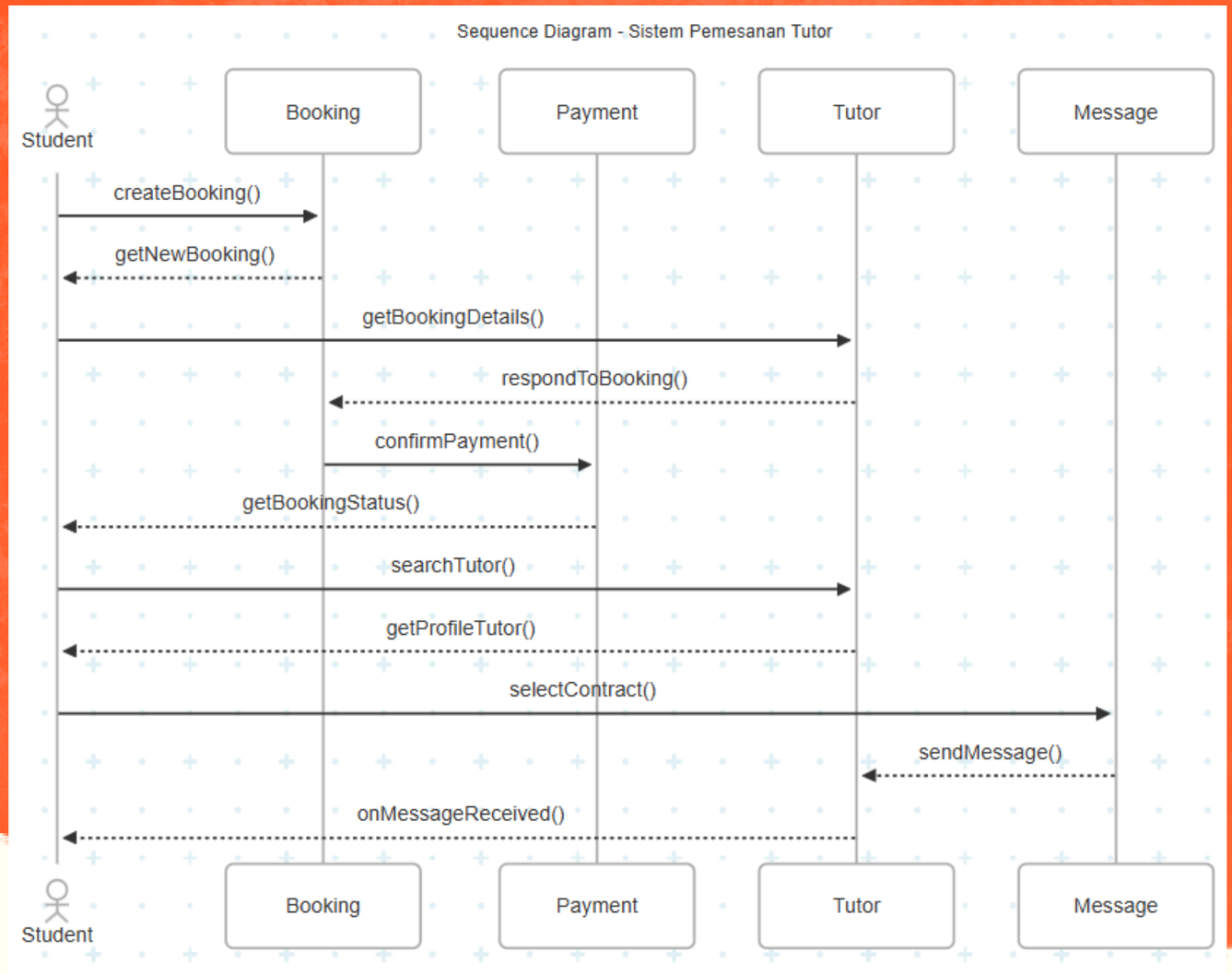
INITIAL COMPONENT ARCHITECTURE



COMPONENT INTERACTION

Tahapan ini bertujuan untuk menggambarkan interaksi dan komunikasi antar komponen dalam sistem secara terkoordinasi.

SEQUENCE DIAGRAM



COMPONENT SPECIFICATION

Tahapan ini bertujuan untuk mendefinisikan spesifikasi antarmuka dan atribut setiap komponen secara lengkap.

IUSERMGT

```
context IUserMgt::registerUser(name: String, email: String, password: String, role: String): User
pre:
  not User.allInstances()->exists(user | user.email = email) and
  email.matches('[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}') and
  password.size() >= 8 and
  (role = 'Student' or role = 'Tutor')
post:
  User.allInstances()->exists(user |
    user.name = name and
    user.email = email and
    user.password = password and
    user.role = role and
    result = user
  )
```

```
context IUserMgt::loginUser(email: String, password: String): User
pre:
  email <> '' and password <> ''
post:
  let matchedUsers: Set(User) = User.allInstances()->select(user |
    user.email = email and user.password = password
  ) in
  matchedUsers->size() = 1 and result = matchedUsers->any(true)
```

```
context IUserMgt::getUserById(userId: String): User
pre:
  userId <> ''
post:
  let users: Set(User) = User.allInstances()->select(user | user.userId = userId) in
  users->size() = 1 and result = users->any(true)
```

```
context IUserMgt::updateUserInfo(userId: String, name: String, email: String, password: String): Boolean
pre:
  User.allInstances()->exists(user | user.userId = userId) and
  (email <> User.allInstances()->select(user | user.userId = userId).email->any(true)
   implies not User.allInstances()->exists(user | user.userId <> userId and user.email = email)) and
  email.matches('[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}') and
  (password <> '' implies password.size() >= 8)
post:
  let user: User = User.allInstances()->select(user | user.userId = userId)->any(true) in
  user.name = name and
  user.email = email and
  (password <> '' implies user.password = password) and
  result = true
```

```
context IUserMgt::getUsersByRole(role: String): Set(User)
pre:
  role = 'Student' or role = 'Tutor'
post:
  result = User.allInstances()->select(user | user.role = role)
```

```
context IUserMgt::deleteUser(userId: String): Boolean
pre:
  User.allInstances()->exists(user | user.userId = userId)
post:
  not User.allInstances()->exists(user | user.userId = userId) and result = true
```

```
context IUserMgt::searchTutorsByName(nameQuery: String): Set(User)
pre:
  nameQuery <> ''
post:
  result = User.allInstances()->select(user |
    user.role = 'Tutor' and
    user.name.toLower().indexOf(nameQuery.toLower()) >= 0)
```

IBOOKINGMGT

```
context IBookingMgt::createBooking(studentId: String, tutorId: String, dateTime: DateTime): Booking
pre:
  User.allInstances()->exists(user | user.userId = studentId and user.role = 'Student') and
  User.allInstances()->exists(user | user.userId = tutorId and user.role = 'Tutor') and
  dateTime > DateTime::now() and
  not Booking.allInstances()->exists(booking |
    booking.tutorId = tutorId and
    booking.dateTime = dateTime and
    booking.status <> 'Rejected'
  )
post:
  Booking.allInstances()->exists(booking |
    booking.studentId = studentId and
    booking.tutorId = tutorId and
    booking.dateTime = dateTime and
    booking.status = 'Pending' and
    booking.paymentStatus = 'Unpaid' and
    result = booking
  )
```

```
context IBookingMgt::getBookingById(bookingId: String): Booking
pre:
  bookingId <> ''
post:
  let bookings: Set(Booking) = Booking.allInstances()->select(booking | booking.bookingId = bookingId) in
  bookings->size() = 1 and result = bookings->any(true)
```

```
context IBookingMgt::updateBookingStatus(bookingId: String, status: String): Boolean
pre:
  Booking.allInstances()->exists(booking | booking.bookingId = bookingId) and
  (status = 'Pending' or status = 'Accepted' or status = 'Rejected')
post:
  let booking: Booking = Booking.allInstances()->select(booking | booking.bookingId = bookingId)->any(true) in
  booking.status = status and result = true
```

```
context IBookingMgt::updatePaymentStatus(bookingId: String, paymentStatus: String): Boolean
pre:
  Booking.allInstances()->exists(booking | booking.bookingId = bookingId) and
  (paymentStatus = 'Paid' or paymentStatus = 'Unpaid') and
  (paymentStatus = 'Paid' implies
    Booking.allInstances()->select(booking | booking.bookingId = bookingId)->any(true).status <> 'Rejected')
post:
  let booking: Booking = Booking.allInstances()->select(booking | booking.bookingId = bookingId)->any(true) in
  booking.paymentStatus = paymentStatus and result = true
```

```
context IBookingMgt::getBookingsByStudent(studentId: String): Set(Booking)
pre:
  User.allInstances()->exists(user | user.userId = studentId and user.role = 'Student')
post:
  result = Booking.allInstances()->select(booking | booking.studentId = studentId)
```

```
context IBookingMgt::getBookingsByDateRange(startDate: DateTime, endDate: DateTime): Set(Booking)
pre:
  startDate <= endDate
post:
  result = Booking.allInstances()->select(booking |
    booking.dateTime >= startDate and
    booking.dateTime <= endDate
  )
```

```
context IBookingMgt::cancelBooking(bookingId: String): Boolean
pre:
  Booking.allInstances()->exists(booking | booking.bookingId = bookingId) and
  Booking.allInstances()->select(booking | booking.bookingId = bookingId)->any(true).dateTime > DateTime::now()
post:
  let booking: Booking = Booking.allInstances()->select(booking | booking.bookingId = bookingId)->any(true) in
  booking.status = 'Rejected' and result = true
```

IMESSAGEMGT

```
context IMessageMgt::sendMessage(fromUserId: String, toUserId: String, text: String): Message
pre:
  User.allInstances()->exists(user | user.userId = fromUserId) and
  User.allInstances()->exists(user | user.userId = toUserId) and
  text <> '' and
  fromUserId <> toUserId
post:
  Message.allInstances()->exists(message |
    message.fromUserId = fromUserId and
    message.toUserId = toUserId and
    message.text = text and
    message.timestamp.isApprox(DateTime::now()) and
    result = message
  )
```

```
context IMessageMgt::getMessagesSentByUser(userId: String): Set(Message)
pre:
  User.allInstances()->exists(user | user.userId = userId)
post:
  result = Message.allInstances()->select(message | message.fromUserId = userId)
```

```
context IMessageMgt::getMessagesReceivedByUser(userId: String): Set(Message)
pre:
  User.allInstances()->exists(user | user.userId = userId)
post:
  result = Message.allInstances()->select(message | message.toUserId = userId)
```

```
context IMessageMgt::deleteMessage(messageId: String): Boolean
pre:
  Message.allInstances()->exists(message | message.messageId = messageId)
post:
  not Message.allInstances()->exists(message | message.messageId = messageId) and
  result = true
```