Data layer

- Data layer was used to specify entities, create repositories, and provide the database. We used object-oriented database.
- Entity classes helped us to easily manage and operate on the data objects. The challenge we faced was how to implement actors of the project in such a way, that every of the actors could have different functionalities and characteristics but be considered still as a user - an entity with assigned role, username, email and password which are used for app security.
- Solution for this, was creating an abstract class AppUser from which the actors inherit fields mentioned above, and still have information and functionalities that are specific to their role only.

Data layer

```kotlin
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class AppUser(
    @Id @GeneratedValue
    open val id: Long = 0,
    open val username: String = "",
    open val email: String = "",
    open val password: String = "",
    @ManyToOne
    open val role: Role
) {}
```

```kotlin
@Entity
data class Driver(
    override val id: Long,
    override val username: String,
    override val email: String,
    override val password: String,
    override val role: Role,
    @OneToOne
    var truck: Truck?,
    @OneToMany
    val packages:MutableList<Package>
): AppUser(id, username, email, password, role){
    Mirson99
    override fun hashCode(): Int = id.toInt()
    Mirson99
    override fun toString(): String ="Driver(id=${id}, name=${username})"

        }
```

```kotlin
@Entity
data class HubWorker(
    override val id: Long,
    override val username: String,
    override val email: String,
    override val password: String,
    override val role: Role,
    @ManyToOne
    val hub:Hub
)  : AppUser(id, username, email, password, role)
```

```kotlin
@Entity
data class Role (
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    val id: Long,
    val name: String,

    @OneToMany(mappedBy = "role")
    val users: MutableList<AppUser>?,
) {
    Mateusz Zieleziński
    override fun toString(): String ="${name}"
}
```

Service layer

- In the service layer we created services for each main endpoints, that used the functions defined in the data layer and helped to communicate between data repositories which store data access objects and their representatives in the front end.
- In the services we allow only given roles to access the functions to match the application logic and requirements.

```
@Service
class HubService (val hubs:HubRepository){

    Antoni Lasik +1 *
    @RolesAllowed("HUBWORKER", "MANAGER")
    fun getAll()=hubs.findAll()
```

Presentation layer

- In presentation layer we have defined the controllers for Clients, Trucks, Hubs, Packages and Messages. In each of the API we used functions from corresponding services to map the DAO into DTO.
- This layer was also responsible for controlling and obeying security rules, such as allowing given user to see his, and only his packages, or massages. Example bellow.

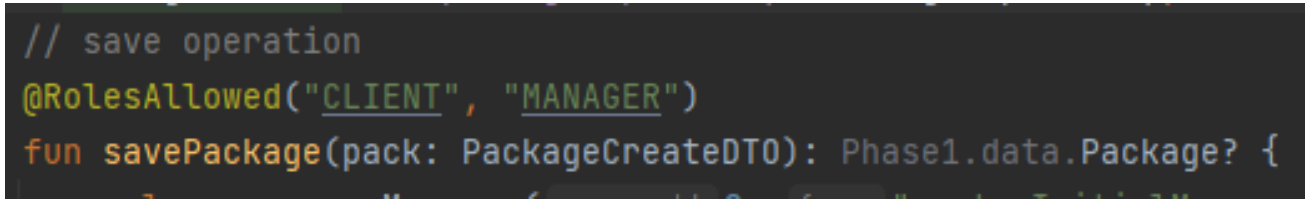
```
@PreAuthorize(CanGetClientPackages.condition)
annotation class CanGetClientPackages {
    Mirson99 +1
    companion object {
        @Language("SpEL")
        const val condition = "(hasRole('CLIENT') AND @mySecurityService.myPackage(principal,#id))" +
                "OR hasRole('MANAGER')"
    }
}
```

User interface layer

- Implemented using typescript with redux (for easy state storage) and react (for quick and efficient user interface creation).
- This layer provides functionalities specified in the user stories. All data needed for interface is provided by presentation layer obtained on requests from client side through API to which client side is connected by proxy.

Backend security rules

- A client can create a shipment.
- A manager can create a shipment (on behalf of any user)



Backend security rules

- A client can track the status and history of their shipments.
- A manager can track the status of any package.

```
package Phase1.configuration.securityRules
import ...

@PreAuthorize(ClientCanGetPackagesInfo.condition)
annotation class ClientCanGetPackagesInfo{
    companion object {
        @Language("SpEL")
        const val condition = "(hasRole('CLIENT') AND @mySecurityService.myUsername(principal,#username))" +
                "OR hasRole('MANAGER')"
    }
}
```

```
@GetMapping("/{username}/packages")
@ClientCanGetPackagesInfo
fun getAllClientPackages(@PathVariable username: String):Collection<PackageClientDto>
```

Backend security rules

- A client can also cancel a shipment if it is under the status Awaiting entry.

```
@PreAuthorize(CanCancelMyPackage.condition)
annotation class CanCancelMyPackage {
    companion object {
        @Language("SpEL")
        const val condition = "hasRole('CLIENT') AND @mySecurityService.myPackage(principal,#id) AND @mySecurityService.packageStatusIsAwaitingEntry(principal, #id)"
    }
}
```

```
@DeleteMapping("{id}")
@CanCancelMyPackage
fun deleteOne(@PathVariable id:Long):Unit
```

Backend security rules

- A client can only see the status of a shipment, and the history of the states of a shipment.

```
@GetMapping("/{id}/statusAndHistory")
@CanGetClientPackages
fun getPackageStatusAndHistory(@PathVariable id:Long): PackageClientDto
```

Backend security rules

- A driver can only change the status of a shipment to in transit or delivered.
- A hub worker can only change the status of a shipment to in storage.
- A manager can change the status of a shipment to any of the possible states.

```
@PreAuthorize(CanUpdatePackage.condition)
annotation class CanUpdatePackage {
    companion object {
        @Language("SpEL")
        const val condition = "(hasRole('DRIVER') AND @mySecurityService.statusIsInTransitOrDelivered(principal, #status)) OR" +
                "(hasRole('HUBWORKER') AND @mySecurityService.statusIsInStorage(principal, #status)) OR" +
                " (hasRole('MANAGER'))"
    }
}
```

```
@PutMapping("/{id}/status")
@CanUpdatePackage
fun updateStatus(@PathVariable id:Long, @RequestParam status: Status?)
```

Backend security rules

- A manager can also change the destination of a shipment.

```
@RolesAllowed("MANAGER")
fun updatePackageDestination(packageId: Long, destination : String?){
    val packageDB: Package = packageRepository.findById(packageId).get()

    if (destination != null) {
        packageDB.putDestination(destination)
    }

    packageRepository.save(packageDB)
    }
```

```
@PutMapping("/{id}/destination")
fun updateDestination(@PathVariable id:Long, @RequestParam destination: String?)
```

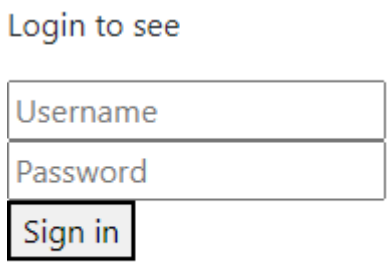
Backend security rules

- No one can edit or delete messages from a mailbox

```
@RequestMapping("api/messages")
interface MessagesAPI {

    @GetMapping("")
    fun getAll():Collection<MessageDTO>

    @PostMapping("")
    fun addOne(@RequestBody message:MessageCreateDTO):Unit

    @GetMapping("{id}")
    fun getOne(@PathVariable id:Long): MessageDTO

//    @DeleteMapping("{id}")
//    fun delete(@PathVariable id:Long):Unit
```

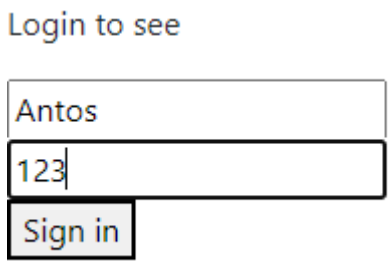
Requirements of the Client Interface

- As an anonymous user, I want access the application and see the main page of the application.



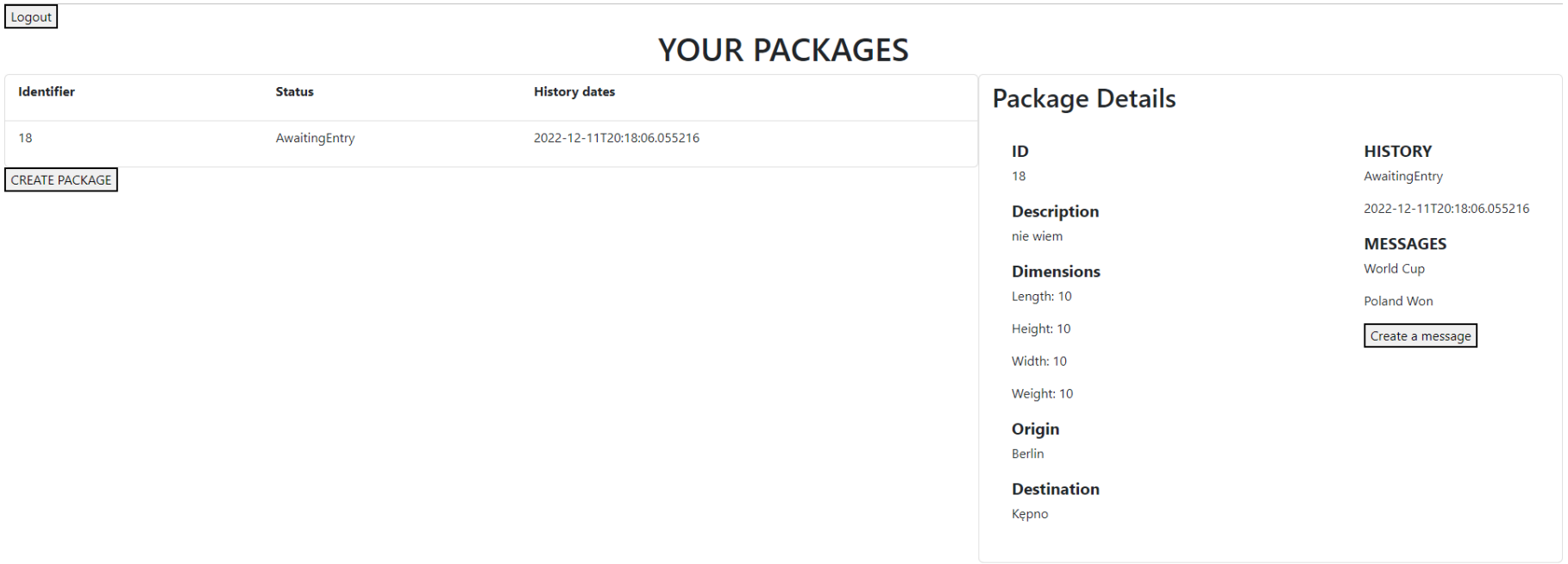
Requirements of the Client Interface

- As an anonymous user, I want to type in my username and password in order to sign in.



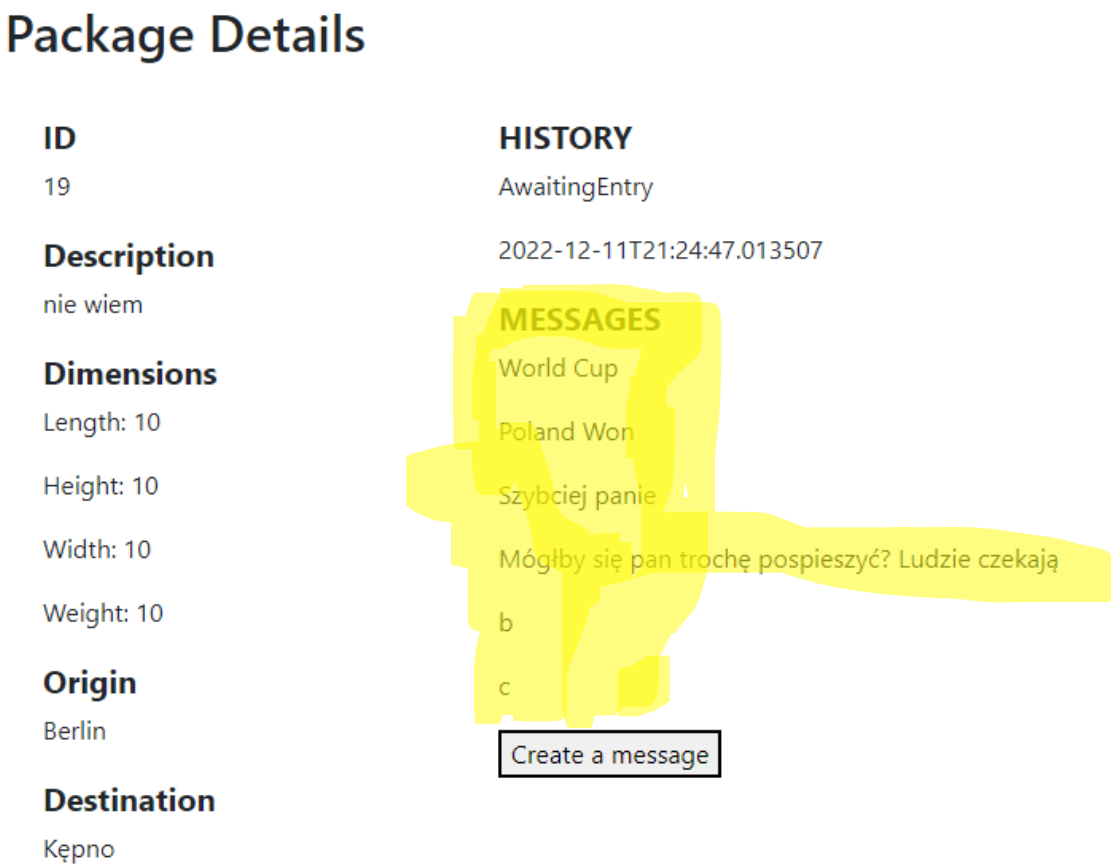
Requirements of the Client Interface

- As a signed-in client, I want to access the homepage and see a list with my shipments present and past, with dates, identification, and status.



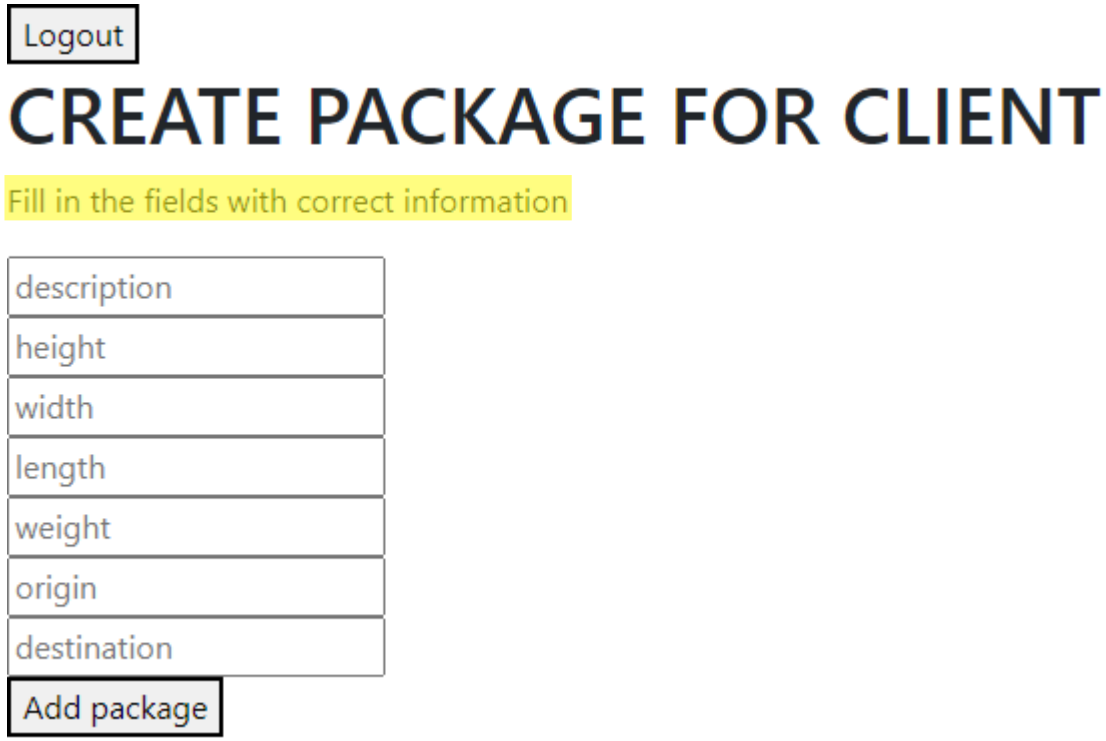
Requirements of the Client Interface

- As a signed-in client, I want to access the homepage and see a list with my messages.



Requirements of the Client Interface

- As a signed-in client, I want to access the homepage and seeing how to create a new shipment.

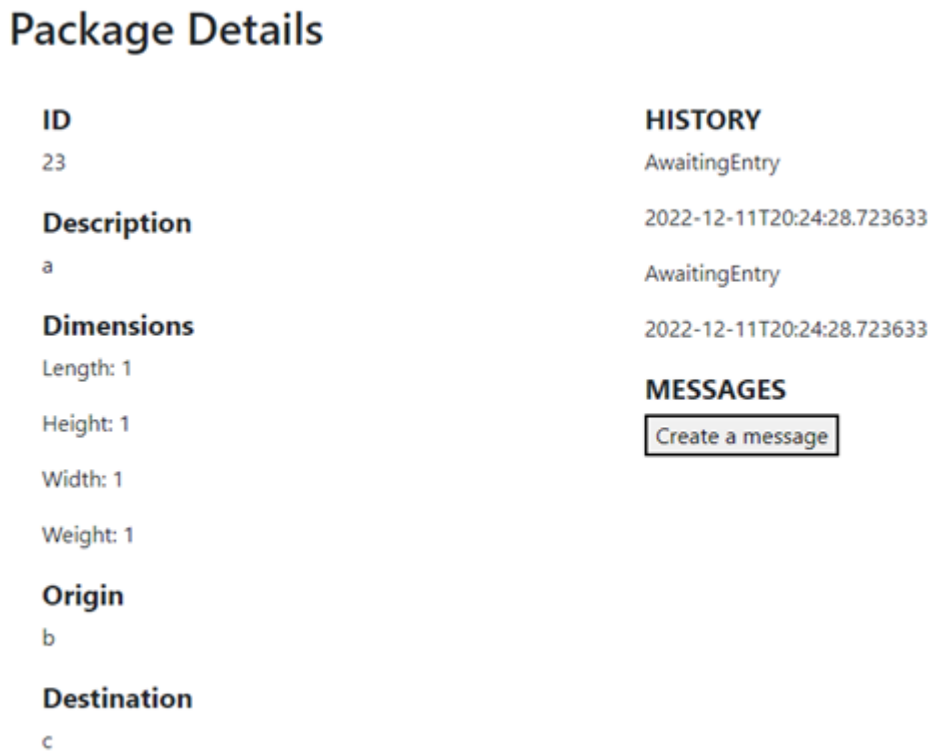


Requirements of the Client Interface

- As a signed-in client, I want to introduce the details of a package to create a new shipment and seeing it in the list of shipments in the status "Awaiting entry".

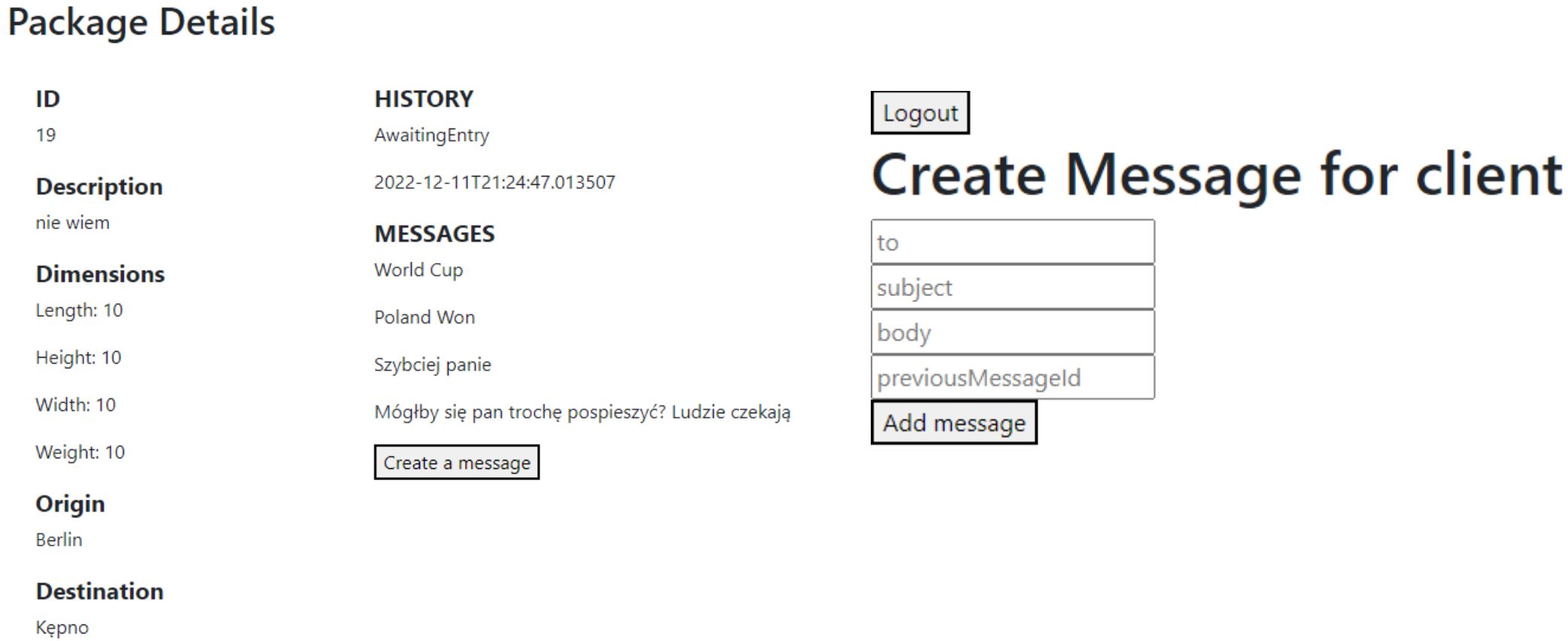
Requirements of the Client Interface

- As a signed-in client, I want to access the list of shipments to select a shipment and seeing the details of a shipment including history, status and messages.



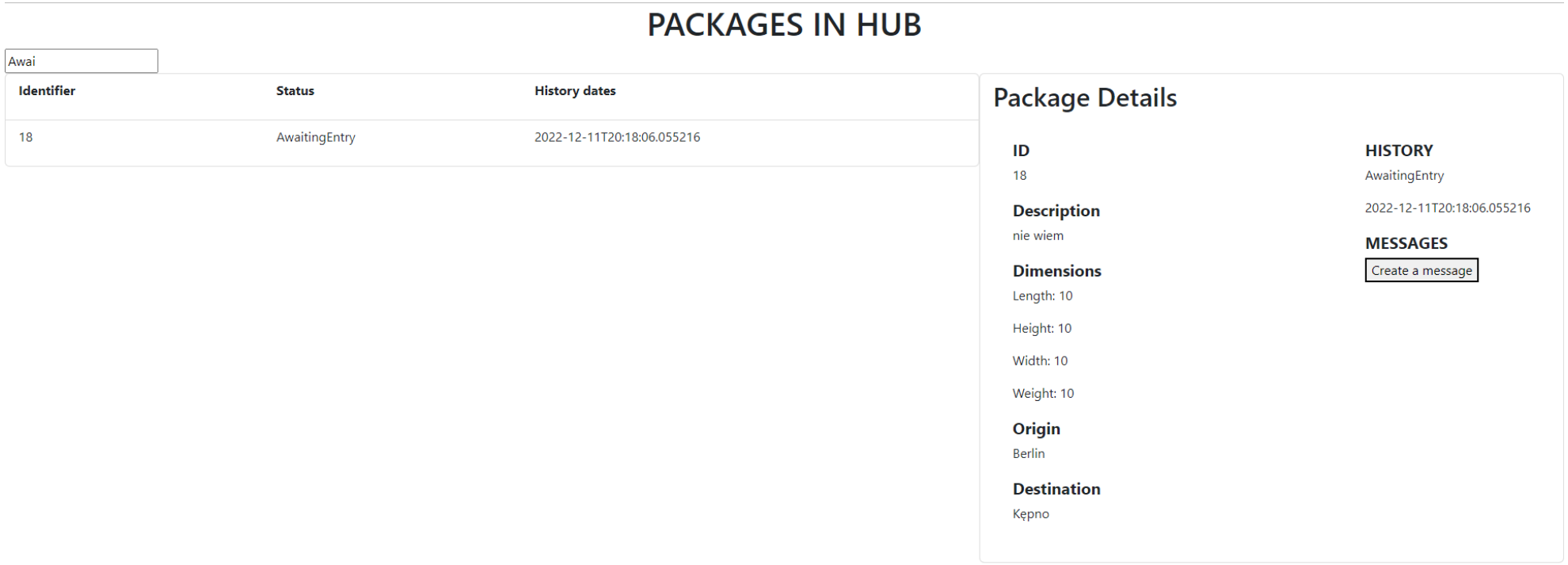
Requirements of the Client Interface

- As a signed-in client, I want to access the details of a shipment to send a message related to the package and seeing the message in the message list.



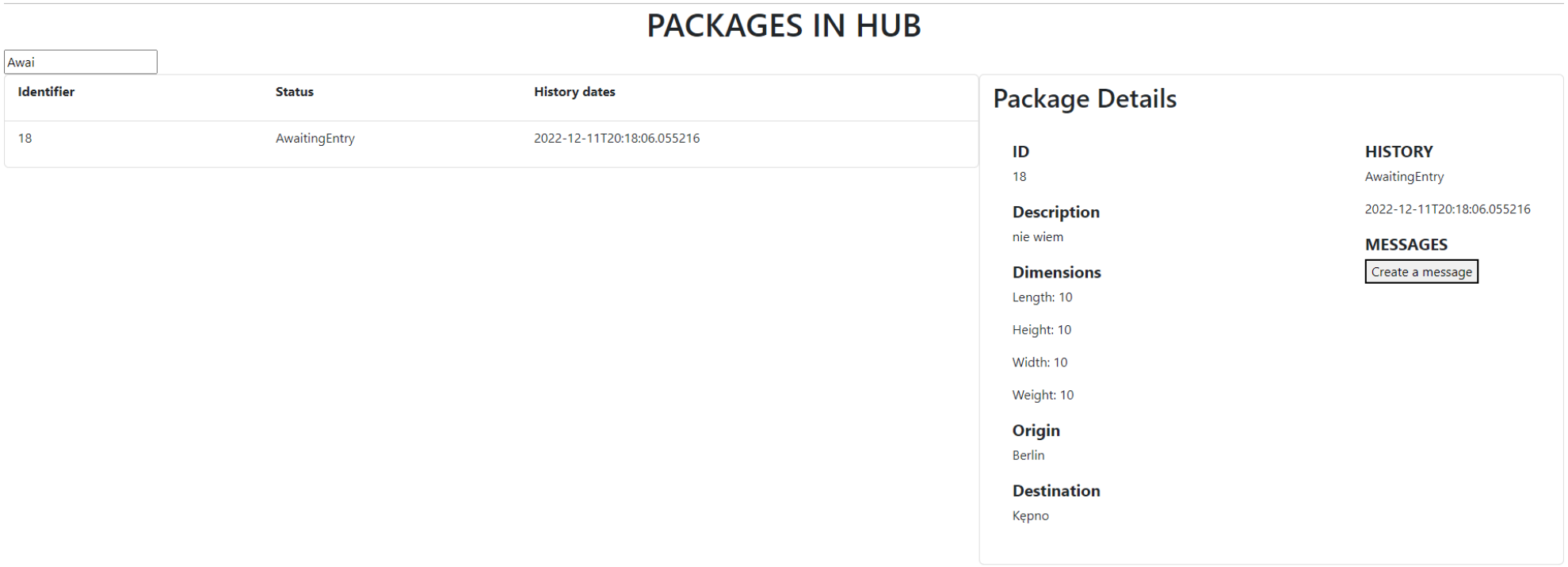
Requirements of the Client Interface

- As a signed-in warehouse worker, I want to see the list of packages in my warehouse to filter the list by status, destination and date and see the filtered list of packages.
- Here we just have the status filtering.



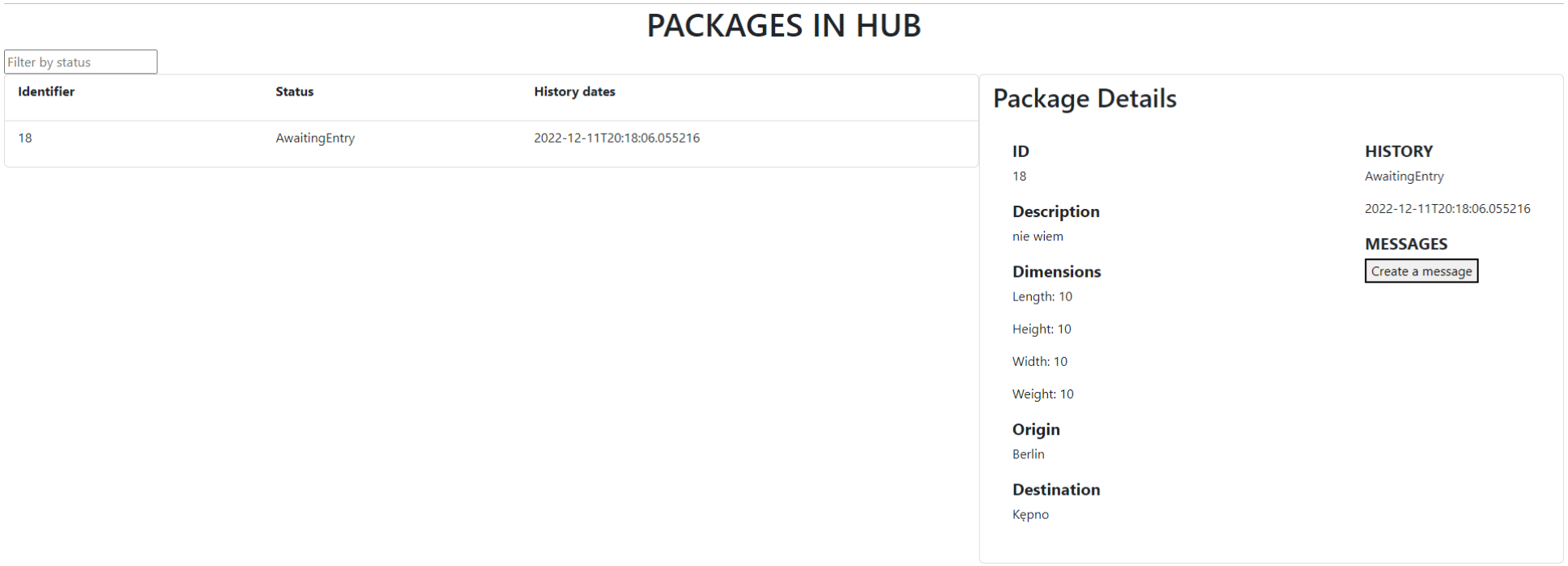
Requirements of the Client Interface

- As a signed-in warehouse worker, I want to see the filtered list of packages in my warehouse to select a package and see the details of the package.



Requirements of the Client Interface

- As a signed-in warehouse worker, I want to access the details of a package to change its status and see the changes in the list of packages.
- Here we didn’t manage to add the status change functionality.

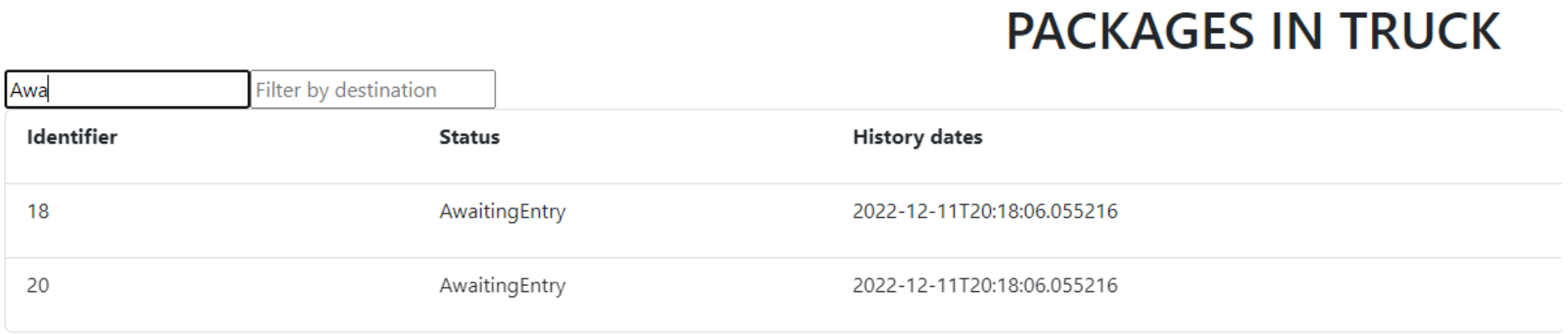


Requirements of the Client Interface

- As a signed-in warehouse worker, I want to access the details of a package to assign it to a truck and see the changes in the list of packages of the truck.
- We didn’t manage to fulfill this rule.

Requirements of the Client Interface

- As a signed-in truck driver, I want to I want to see the list of packages in my truck to filter the list by destination, and date and see the filtered list of packages.
- We don’t have filtering based on date implemented.

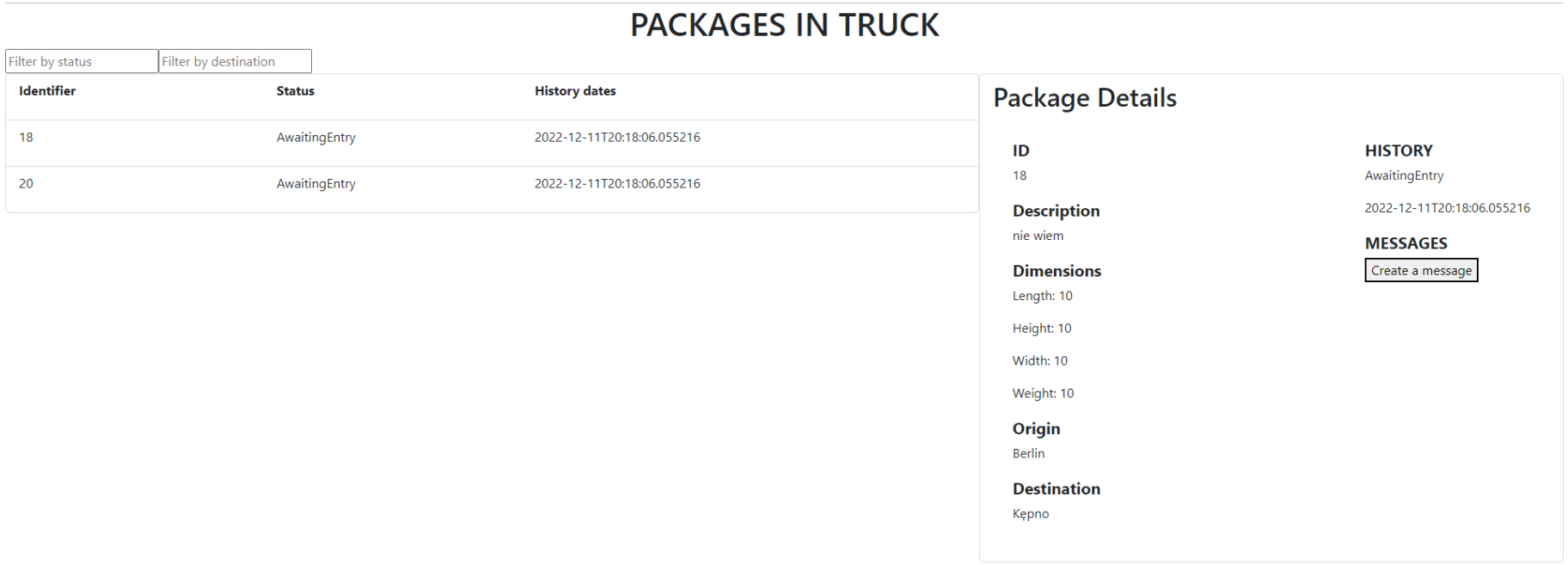


Requirements of the Client Interface

- As a signed-in truck driver, I want to access the details of a package to assign it to a warehouse and see the changes in the list of packages of the warehouse.
- We didn’t manage to fulfill this rule.

Requirements of the Client Interface

- As a signed-in truck driver, I want to access the details of a package to change its status and see the changes in the list of packages.
- Here we didn’t manage to add the status change functionality.



Requirements of the Client Interface

- As a signed-in user, I want to sign-out and see the main page of the application.