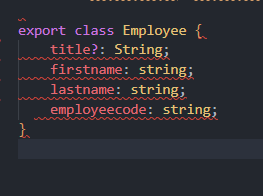
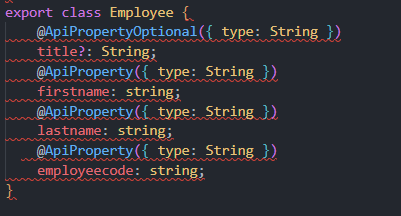
# Payroll Company / Pay packet notes S Nesbitt

We want to know if this project was a lot bigger, what tech, what design principles, who would you need, what decisions need to be formalised, what stakeholders need to be investigated,

**API - models**

1. Open API and investigate
2. Discover it’s running swagger - see if schemes/models are defined
3. Only “component” – so firstly I’d need “devs” to ensure appropriate types are enabled for each API. I believe there is a nest swagger plugin that changes the code to simple models



1. Then generate types on the UI – both as part of the dev pipeline (npm command) and part of the build/test/deploy pathway
2. The API models etc all need appropriate comments for the swagger definition
3. A live postman collection would be useful with examples for UI dev’s

**API – coding standards / project**

1. Would have to have a discussion around whether the to main current structure of [component -> specs, controllers, interace, module, service] or [interfaces, services, controllers, modules]
2. Would need to discuss whether NEST.js is the best option, or whether there are other / better alternatives. Discussion around learning curve versus simply express node.js project with appropriate folder structure/libraries/documentation
3. Consideration around complex API’s that need to use models across multiple different types
4. Consider GRAPHQL and pro’s / con’s
5. Technology assessment – determine what tech is upcoming and relevant in the API space to ensure not using old tech. But balance with “newer” does not mean “better”
6. “Component” is a terrible name for an API
7. API for multiple – end with an (s) e.g. employees possibly
8. Project doesn’t have CORS enabled
9. employeeCode should be mapped to “id”
10. Employees returns “employeecode” and paypacket returns “employeeCode” – inconsistent naming
11. Paypacket status should be enum or number not string and ids aren’t sequential

**Other concerns**

1. Security – if it’s public then needs security consideration / audits, rate limiting, authentication and sessions (JWT)
2. Documentation – agreements on API status codes, how errors should be transmitted, sanitization, naming conventions, typescript definitions, coding standards all documented
3. Paypacket edit – there is no edit. It just adds a new paypacket component which seems flawed.
4. Paypacket – there is no way to create a new paypacket. Need to define requirements
5. Need to determine proper requirements here, if we need to insert a new component for auditing but logic needs to be considered and documented before writing UIthat’s fine
6. Employees API could include aggregated stats
7. Probably need API for stats, then need to consider best way to compile this (caching, daily generation flat files)

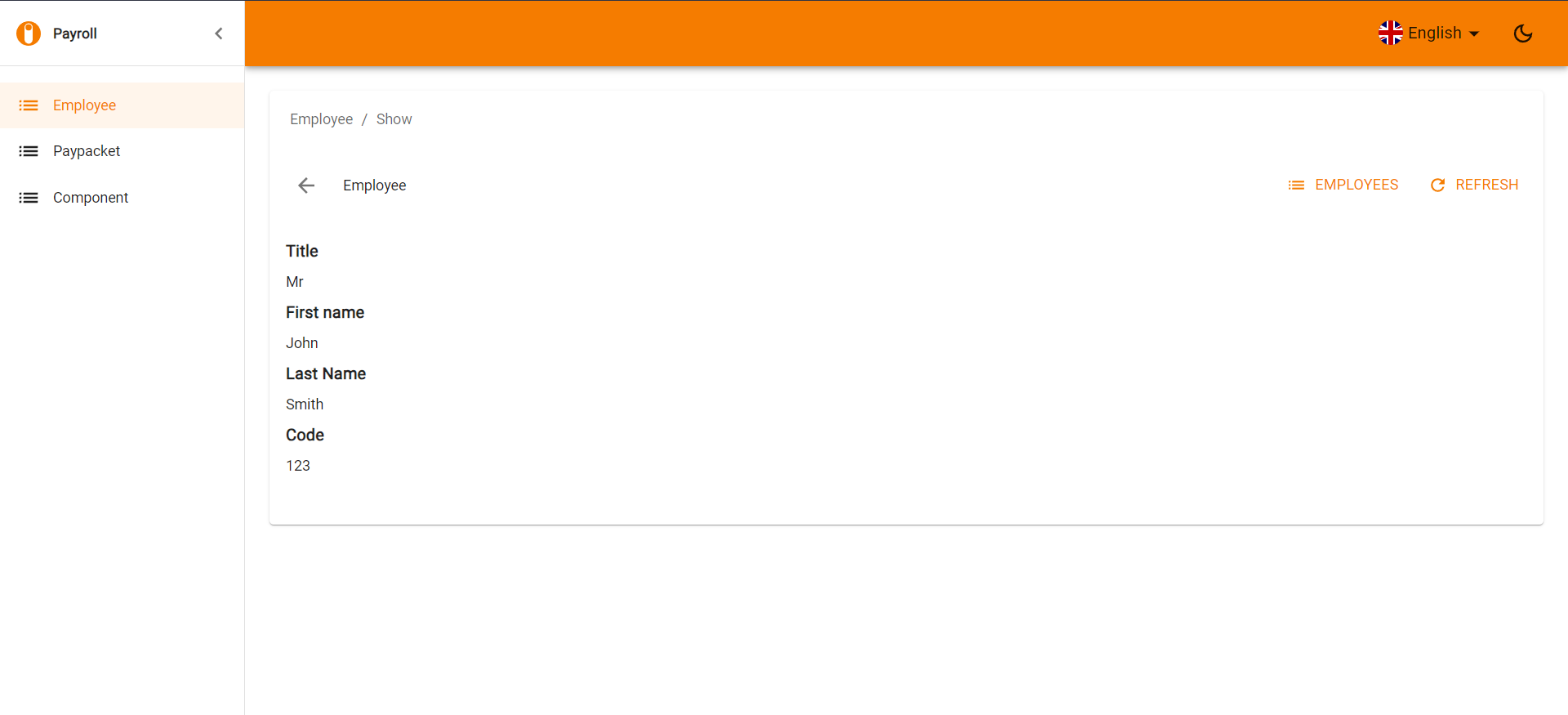
**Refine.dev**

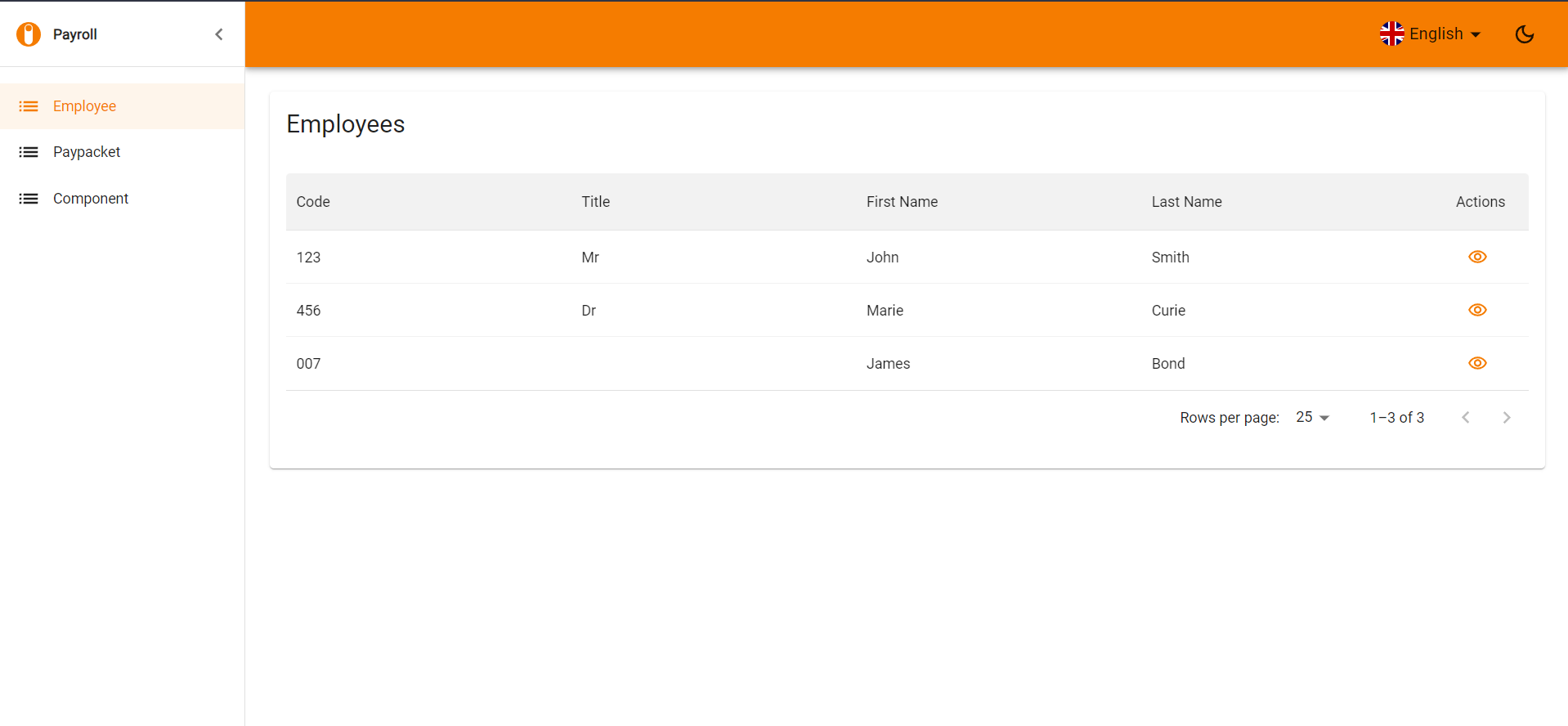
1. Next.js, Material UI, Nest Integration, Ignore Auth for now
2. Date Library
3. UI Library
4. Translations needed?
5. Security concenrs?
6. Responsive/performance limitations
7. Browser specs / requirements
8. Accessibility

**Project UI**

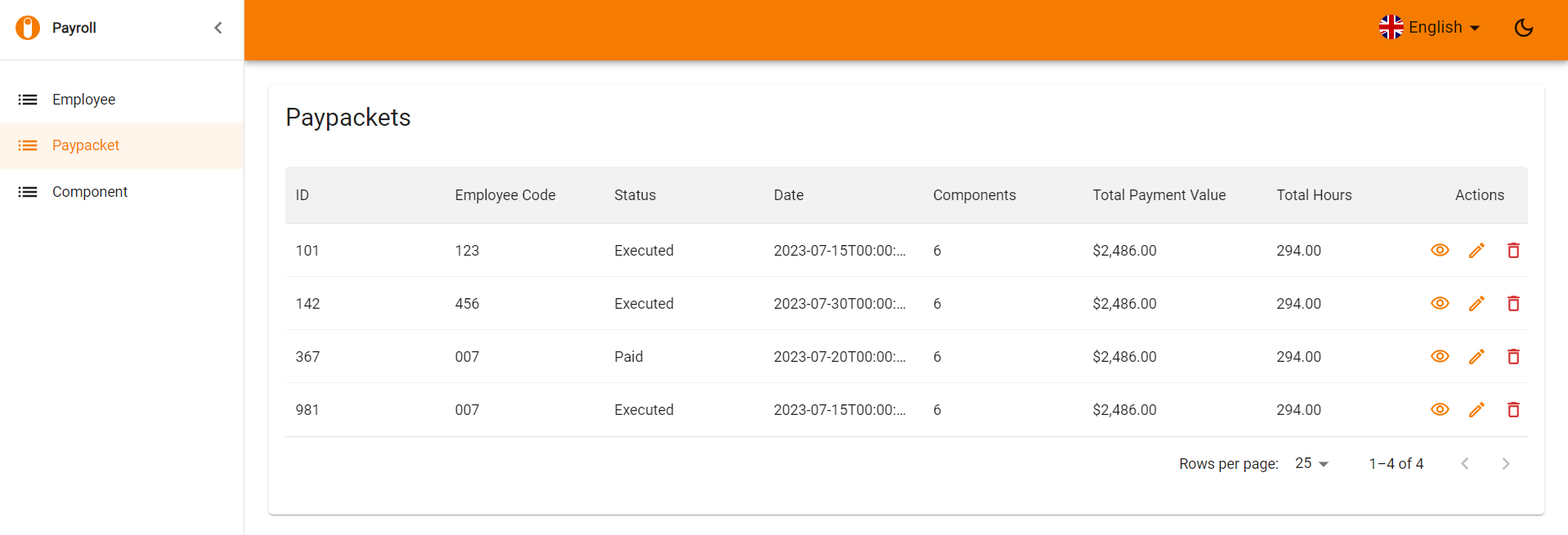
1. Come up with best UI to handle mapping / editing between relationship data
2. Need to change table to have better filters based on relationship data
3. When picking a relationship item in create/edit needs autocomplete or drop downs (i.e. enum drop down)
4. Comprehensive validation needs to be documented
5. Unit testing and Automated testing TBC
6. Typescript models to be defined or auto generated from swagger definition
7. AA accessibility to be tested and confirmed
8. Repsonsiveness to be defined and tests at < tablet, < desktop, < large desktop
9. Performance specifications (payload needs to be under X mb, < 2 second load time)
10. Include appropriate loaders and “Skeleton” for visual feedback
11. Consider walkthrough / documention
12. Storybook
13. Ensure better filters / search

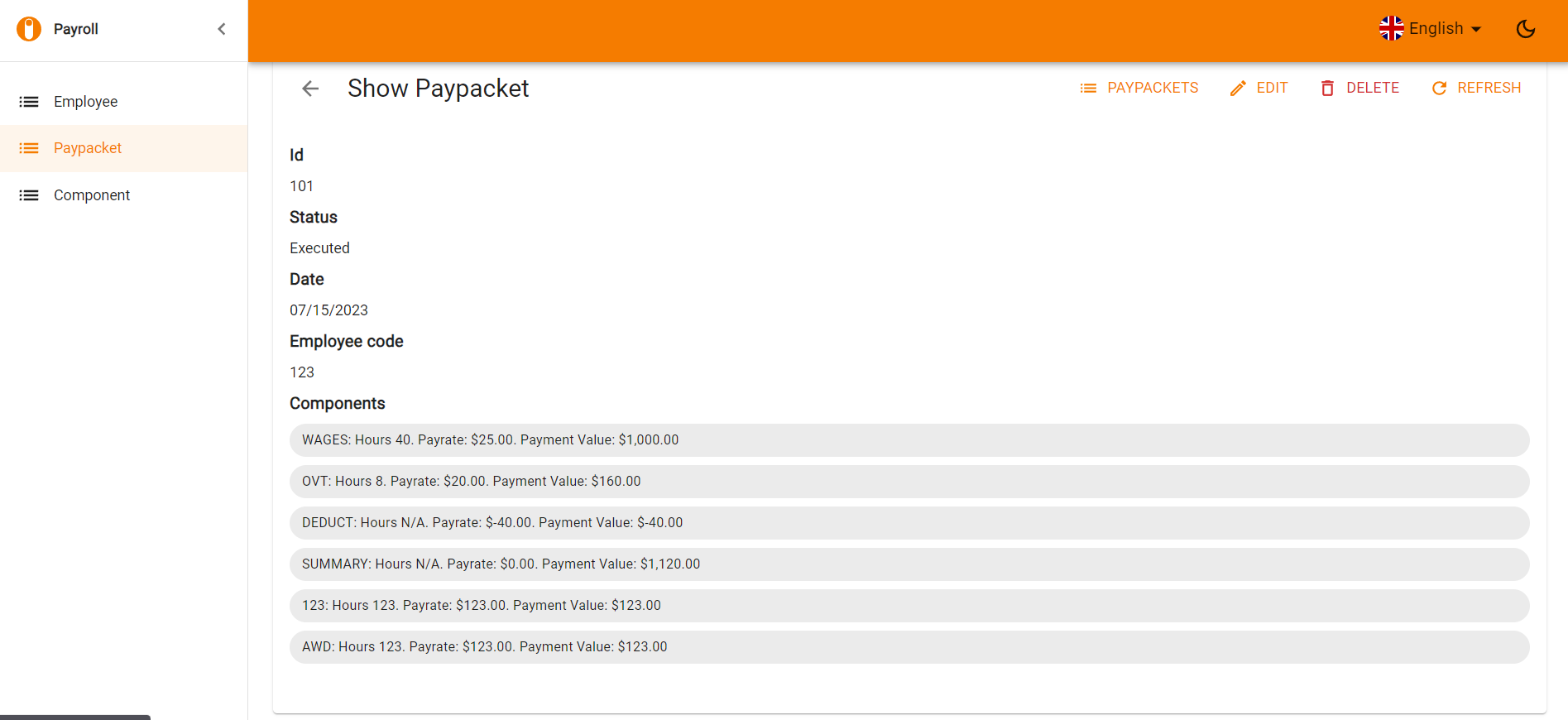
**Employees View / Details Only**

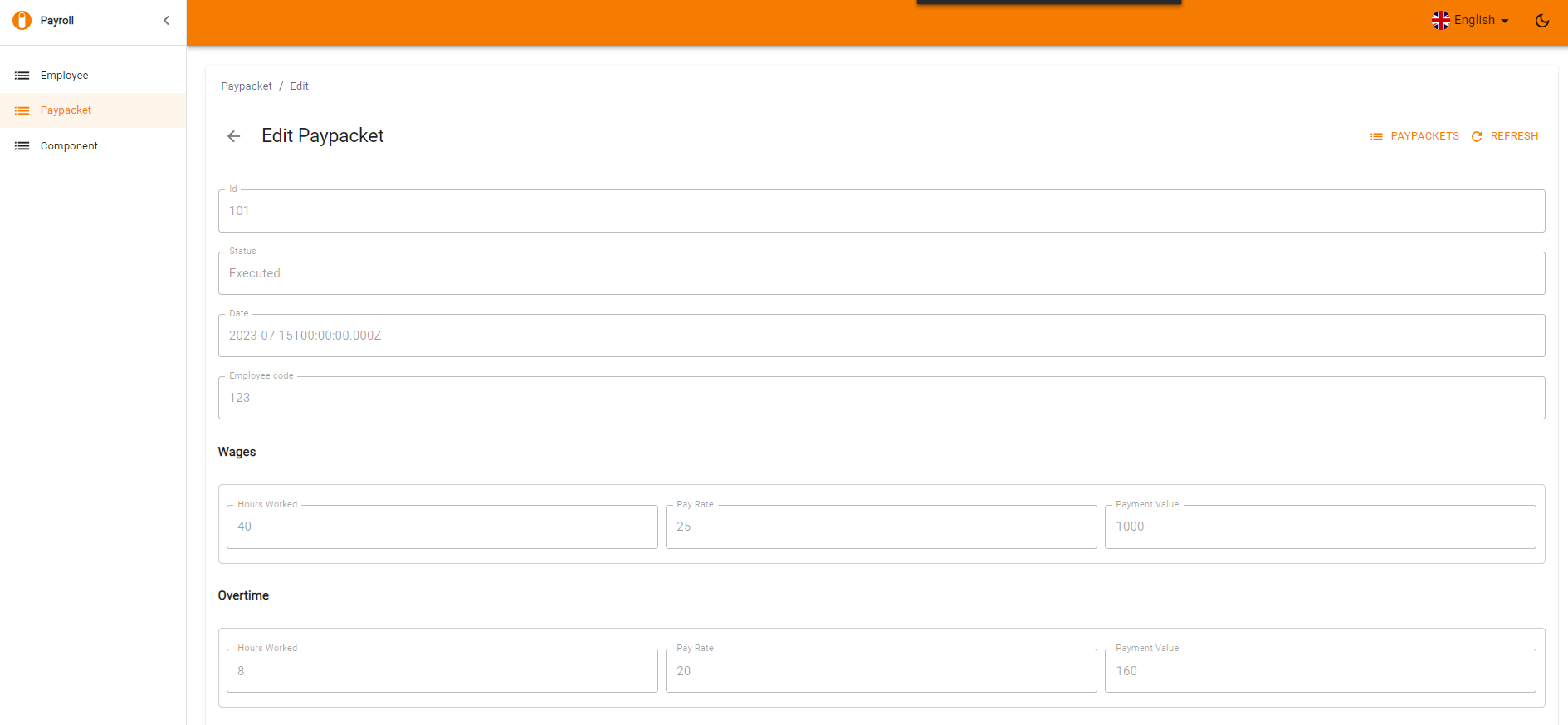


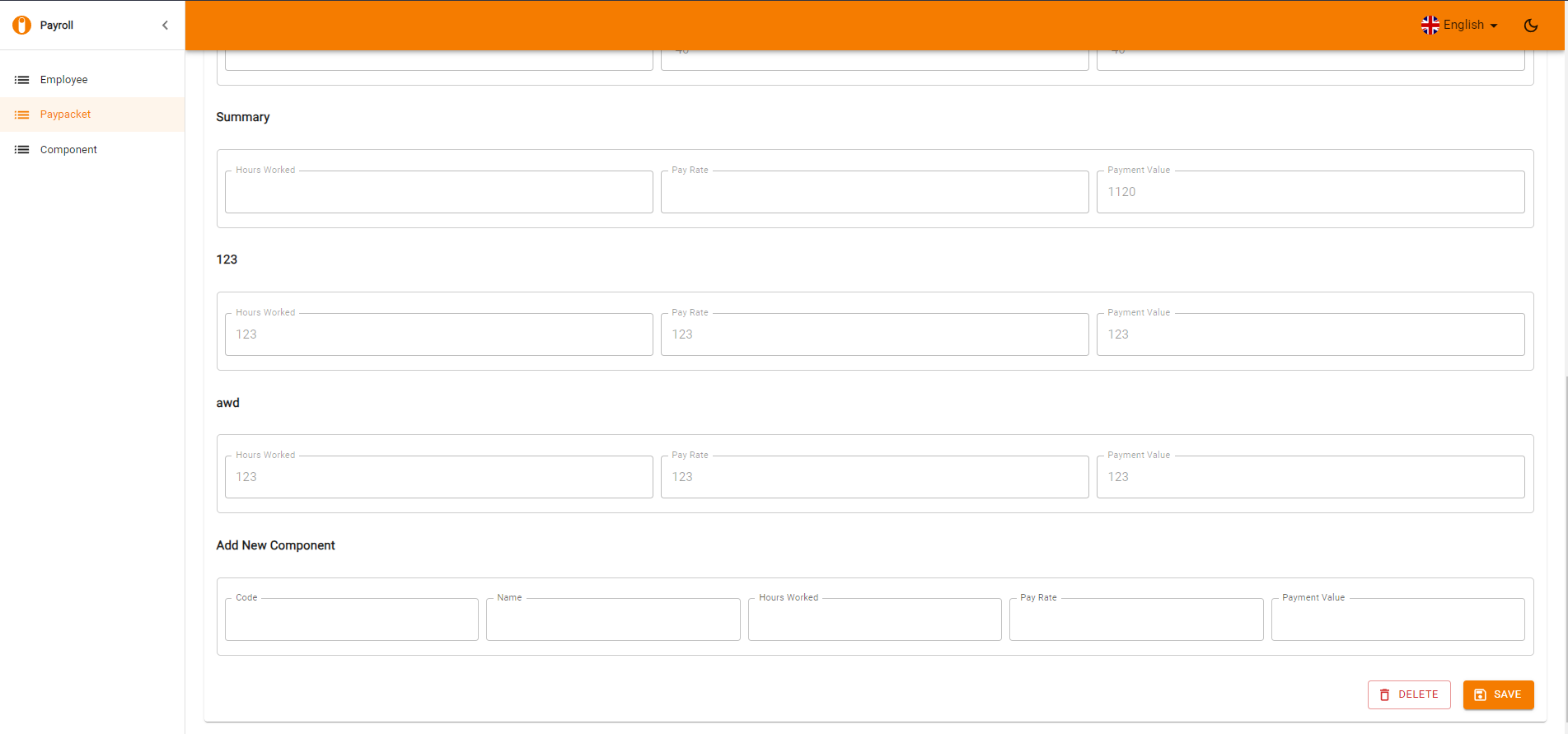


Paypacket









**OTHER**

1. More to be included
   1. GST
   2. Sick days
   3. Late Days
   4. Reconcilation Reports
   5. Reversals
   6. Notes
   7. Attachments - timesheets
   8. Assignee
   9. Case Manager
   10. Due Date
   11. Costs – Currency conversion

Realistically depending on the scope you’d need a PO, BA, 3 devs, 1-2 testers, 1 ux/designer, a lot of coffee. Would need to scope out

**New Devs**

Include storybook and go through code examples

Appropriate documentation around coding standards

Weekly/fortnightly demo’s and show cases

Pair programming, work with testers

Include meetings about how we work

Include meetings around retro – important with new devs to have their feedback