Poker Hand Evaluator - Report

How to use

This program takes a five-card poker hand and outputs the best possible poker hand ranking that can be made from those cards.
The user is prompted to enter 5 cards (separated by nothing or any number of spaces)

Each card must include

- A 'value' which is either a capital letter or a single digit, representing the 13 card values A, 2, 3, 4, 5, 6, 7, 8, 9, T (representing 10), J, Q, K
- A 'suit' which is in lower case, either s, h, d, or c, representing Spades, Hearts, Diamonds and Clubs respectively

Any card exceeding 5 will be discarded

Then we are given the list obtained by parsing that string, followed by the best possible poker hand ranking for that hand.

For example, this hand:

... would be represented as `Jh Js 3c 3s 2h`

The output for this input would be:

```
[J♥,J♠,3♣,3♠,2♥]
TwoPair
```

If there are less than 5 cards or any of the cards appear twice (not possible in poker) then it will be an `InvalidHand`
There are some tests for all possible hands which can be seen with `stack test`
The program is run, of course, by `stack run`

Explanations

Cards
The cards are represented by a record data type containing a Value and Suit, which are explained above. This was used because it defines the value and suit functions which are used later. The Values and Suits are in order of high cards.
Hand Parser
Each card is parsed by first reading the value and then the suit and skipping space characters then returning the card of those two.

The values and suits are both parsed by reading them (defined the same way for both using a typeclass), then using `fmap` (`<$>`) on all the possible characters – i.e. if it's a suit character it will be read to a Suit (and same for value). The 'possible characters' are worked out by either valChar or suitChar which just check using `satisfy` if it is an element of the char array defined in Cards
Hand Rankings
Here a new data type for all the rankings is defined the same way as the values and suits. The hand ranking for a card is obtained by checking if it is a royal flush, then checking all of them down to pair, this will give the highest possible hand.

Getting the frequencies of the card, to determine pairs, n of a kind, etc. is done by using a map to efficiently create a list of tuples of values and ints (the value function mentioned above is used here)
For straights, since ace can be either low or high and I have defined ace as high we need to check that *either* all the elements of the sorted value list are in sequence, *or* compare it directly with the aceLowStraight, as we did with the royal straight (ace high) when checking for a royal flush.

Primary Library used : Megaparsec

Personal Experience

I found the library to be a learning curve but once I had read lots of explanations of parsers in general, and how to use megaparsec I found it pleasant and easy to use, despite being rather complicated at first.
I thought the parser was the most difficult bit but most interesting as I learned a lot about functional parsing (something new).

List of resources

Learning for parser:

https://www.youtube.com/watch?v=dDtZLm7HIJs
https://mmhaskell.com/parsing/megaparsec
https://akashagrawal.me/2017/01/19/beginners-guide-to-megaparsec.html
https://markkarpov.com/tutorial/megaparsec.html
https://hasura.io/blog/parser-combinators-walkthrough/
https://en.wikipedia.org/wiki/Parser_combinator
https://serokell.io/blog/parser-combinators-in-haskell
https://stackoverflow.com/questions/14655157/parsing-haskell-custom-data-types

other:
https://en.wikipedia.org/wiki/List_of_poker_hands,
https://en.wikipedia.org/wiki/Playing_cards_in_Unicode
http://learnyouahaskell.com
various lookups on Hoogle and some on zvon.org, which had some function explanations which came up when I looked up on google.
Some errors with imports etc. I looked up on stackoverflow.com
Hackage for some of the packages esp. Megaparsec, tasty
https://stackoverflow.com/questions/26217871/haskell-function-that-tests-if-a-list-has-repeated-duplicate-elements (nub idea)
https://www.vacationlabs.com/haskell/adt.html
https://wiki.haskell.org/GHC/Type_families

Learning for parser:

https://www.youtube.com/watch?v=dDtZLm7HIJs
https://mmhaskell.com/parsing/megaparsec
https://akashagrawal.me/2017/01/19/beginners-guide-to-megaparsec.html
https://markkarpov.com/tutorial/megaparsec.html
https://hasura.io/blog/parser-combinators-walkthrough/
https://en.wikipedia.org/wiki/Parser_combinator
https://serokell.io/blog/parser-combinators-in-haskell
https://stackoverflow.com/questions/14655157/parsing-haskell-custom-data-types