

Übungsblatt 6

Aufgabenlösung

Abgabe: 06.07.2017

6.1 *Hoogle Docs*

Server Die Messages sowie das JSON Encoding und Decoding wurden erweitert. So senden wir in der ClientMessage die Position des Cursors des Clients mit. Außerdem wurde die ServerMessage um sowohl eine Map von Cursors, welche momentan an dem Sheet sind, bei der RemoteEdit-Message erweitert als auch um eine komplett neue Message, welche wir recht eloquent als *HelloGuys* bezeichnet haben. Diese ist dafür zuständig, bei einer neuen Clientverbindung den anderen Clients zu signalisieren, einen neuen Cursor zu ihrer lokalen Version des Sheets hinzuzufügen.

Im Zuge der dritten Aufgabe wurden außerdem *Retain*, *Insert* und *Delete* um *Int* bzw. ein *String*-Feld beim Encoding sowie Decoding erweitert. Für die neuen Messages sowie Anpassungen wurden ebenfalls die bereits vorhandenen En- und Decodings angepasst.

```

9  data ClientMessage = ClientEdit Int TextOperation Cursor
10
11  data ServerMessage = RemoteEdit TextOperation (M.Map Int Cursor)
12                      | Ack
13                      | HelloGuys
14
15  instance FromJSON TextAction where
16    parseJSON (Number x) = do
17      n ← parseJSON (Number x)
18      return $
19        if n < 0 then Delete $ abs n
20        else Retain n
21    parseJSON (String t) = do
22      c ← parseJSON (String t)
23      return $ Insert c
24
25  instance FromJSON TextOperation where
26    parseJSON (Array a) = do
27      ops ← parseJSON (Array a)
28      return $ TextOperation ops
29
30  instance FromJSON ClientMessage where
31    parseJSON (Object o) = ClientEdit
32      <$> o .: "rev"
33      <*> o .: "op"
34      <*> o .: "cursor"
35
36  instance ToJSON TextAction where
37    toJSON (Retain n) = toJSON (n :: Int)
38    toJSON (Delete n) = toJSON (-n :: Int)
39    toJSON (Insert c) = toJSON c
40
41  instance ToJSON TextOperation where
42    toJSON (TextOperation as) = toJSON $ map toJSON as
43
44  instance ToJSON ServerMessage where
45    toJSON Ack = toJSON ("ack" :: String)
46    toJSON HelloGuys = toJSON ("helloguys" :: String)
47    toJSON (RemoteEdit op cursors) = object ["op" .:= toJSON op,
48      "cursors" .:= cursors'] where cursors' = toJSON $ M.elems cursors

```

Da der Server für die konsistente Verwaltung der Cursor zuständig ist, haben wir den ServerState um *cursors* erweitert, welches den Cursor als Int für jeden jeweiligen Client mit seiner id identifiziert in einer Map verwaltet. Dies ist selbstverständlich im anfänglichen ServerState *initialState* eingebaut.

```

21  data ServerState = ServerState {
22    ot :: Server TextOperation String,
23    idGen :: ClientId,
24    clients :: [(ClientId, WS.Connection)],
25    cursors :: M.Map ClientId Cursor
26  }
27
28  type ClientId = Int
29
30  initialState = ServerState {
31    ot = (Server (" " :: [Char]) ([] :: [TextOperation])),
32    idGen = 0,
33    clients = [],
34    cursors = M.empty
35  }

```

Die Verarbeitung einer *ClientEdit*-Nachricht, welche um den Cursor erweitert ist, wurde ebenfalls angepasst. Diese muss nun nämlich zum einen den neuen Cursor so transformieren, dass dieser mit dem aktuellen ServerState konsistent ist (*transform*) und zum anderen werden die bereits vom Server verwalteten Cursor durch die neu vom Client geschickte Operation angepasst. Somit bilden wir die neue Map von Cursors, welche wir nun den anderen Clients zukommen lassen; bis auf den eigenen Cursor des Clients, welcher von diesem selbst verwaltet wird.

```

37 handleSocket :: WS.Connection → ClientId → Int → MVar ServerState → IO ()
38 handleSocket conn id offset server = do
39   — Empfange die Daten vom Client
40   let disconnect err = do
41     putStrLn $ "disconnected: client_" # show id
42     modifyMVar server $ \s →
43       return s { clients = filter ((≠ id). fst) (clients s),
44                 cursors = M.delete id $ cursors s }
45     throw (err :: WS.ConnectionException)
46   msg ← catch (WS.receiveData conn) disconnect
47   case decode msg of
48     Just (ClientEdit rev op cursor) → do
49       print op
50       op' ← modifyMVar server $ \s → do
51         let (op', ot') = appendOperation (ot s) op (offset + rev)
52         let newCursor = foldl (flip transformCursor) cursor $ drop (offset + rev) $ history o ot $ s
53         let newOtherCursors = M.map (transformCursor op) $ cursors s
54         let newCursors = M.insert id newCursor newOtherCursors
55         print newCursors
56         forM_ (clients s) $ \ (cid, client) → do
57           let cursors' = M.filterWithKey (\k _ → cid ≠ k) newCursors
58           if cid ≠ id
59             then do
60               WS.sendTextData client (encode (RemoteEdit op' cursors'))
61               print cursors'
62             else WS.sendTextData client (encode Ack)
63         return (s { ot = ot', cursors = newCursors }, op')
64       handleSocket conn id offset server
65     Nothing → do
66       print msg
67       error "could not decode client message"

```

Verbindet sich ein neuer Client, so wird dessen Cursor zum Cursor in seiner Map hinzugefügt sowie den übrigen Clients eine *HelloGuys*-Message gesandt, um kundzutun, dass sich ein neuer Client connected hat und dessen Cursor dargestellt werden soll.

```

77 socket server pending = do
78   conn ← WS.acceptRequest pending
79   WS.forkPingThread conn 30
80   (id, offset) ← modifyMVar server $ \s → do
81     let id = 1 + idGen s
82     let clients' = (id, conn) : clients s
83     let offset = serverRevision (ot s) - 1
84     let stateOp = serverStateOp (ot s)
85     let cursors' = M.insert id 0 $ cursors s
86     WS.sendTextData conn (encode (RemoteEdit stateOp $ cursors s))
87     forM_ (clients s) $ \ (_, con) →
88       WS.sendTextData con $ encode HelloGuys
89     return (s { idGen = id, clients = clients', cursors = cursors' }, (id, offset))
90   putStrLn $ "connected: client_" # show id
91   handleSocket conn id offset server

```

Analog zu der transform-Funktion der Operationen wurde eine Transformations-Funktion für die Cursor hinzugefügt. Diese verschiebt den jeweiligen Cursor nach rechts, wenn links von diesem etwas Neues hinzugefügt wurde und nach links, wenn links von diesem etwas gelöscht wurde. Das Verschieben ist hierbei angepasst auf die Anzahl der neuen bzw. gelöschten Elemente links vom jeweiligen Cursor. So wird dieser bei zwei neuen Elementen auch um zwei Stellen verschoben.

```

116 type Cursor = Int
117
118 transformCursor :: TextOperation → Cursor → Cursor
119 transformCursor (TextOperation op) c = foldr transf c $ split op c where
120   transf op c = case op of
121     Retain n → c
122     Insert str → c + length str
123     Delete n → c - n
124   split [] _ = []
125   split (x:xs) c
126   | c ≤ 0 = []
127   | otherwise = case x of
128     Retain n →
129       if n ≤ c
130       then x : split xs (c-n)
131       else [Retain c]
132     Delete n →
133       if n ≤ c
134       then x : split xs (c-n)
135       else [Delete c]
136     Insert cs →
137       if length cs ≤ c
138       then x : split xs (c - length cs)
139       else [Insert $ take c cs]

```

Für Aufgabe 3 wurde eine effizientere Darstellung der Retains, Inserts und Delete konstruiert.

```

16 data TextAction = Retain Int
17                 | Insert String
18                 | Delete Int deriving Show

```

Im Zuge dieser Veränderung mussten selbstverständlich auch *noop* und *applyOp* angepasst werden.

```

65 instance Document String TextOperation where
66   noop d
67   | length d == 0 = TextOperation []
68   | otherwise = TextOperation [Retain $ length d]
69   applyOp d (TextOperation as) = apply' as d where
70     apply' [] [] = []
71     apply' (Retain 0 : as) cs = apply' as cs
72     apply' (Retain n : as) (c:cs) = c : apply' (Retain (n-1) : as) cs
73     apply' (Insert cs : as) d = cs ++ apply' as d
74     apply' (Delete 0 : as) cs = apply' as cs
75     apply' (Delete n : as) (.:cs) = apply' (Delete (n-1) : as) cs

```

Die Funktionen *compose* und *transform* wurden ebenfalls angepasst auf die neue Darstellungsform. Da im Grunde dasselbe passiert wie bei der simpleren Repräsentation der Retains, Inserts und Deletes, werden wir diese nicht bis ins kleinste Detail erklären. Das einzig Interessante an der Anpassung ist, dass für jeden Fall, welcher nicht ein beliebiges Element auf linker oder rechter Seite hat, nun überprüft werden muss, ob die linke Anfangsoperationsfolge oder die rechte kleiner ist. Die kleinere wird nun genommen, analog zu den vorherigen Fällen verarbeitet und die größere so gesplittet, dass genau so viele Aktionen wie im kleineren verarbeitet werden und der Rest weiter angeketet an den Rest nun mit den nachfolgenden Folgen des kleineren rekursiv weiterverarbeitet wird.

```

22 instance Operation TextOperation where
23   compose (TextOperation a) (TextOperation b) = TextOperation $ compose' a b where
24     compose' [] [] = []
25     compose' (Delete n : as) bs = Delete n : compose' as bs
26     compose' as (Insert cs : bs) = Insert cs : compose' as bs
27     compose' (Retain n : as) (Retain n' : bs)
28     | n > n' = Retain n' : compose' (Retain (n-n') : as) bs
29     | n < n' = Retain n : compose' as (Retain (n'-n) : bs)
30     | otherwise = Retain n : compose' as bs
31     compose' (Retain n : as) (Delete n' : bs)
32     | n > n' = Delete n' : compose' (Retain (n-n') : as) bs
33     | n < n' = Delete n : compose' as (Retain (n'-n) : bs)
34     | otherwise = Delete n : compose' as bs
35     compose' (Insert cs : as) (Retain n : bs)
36     | n > length cs = Insert cs : compose' as (Retain (n-length cs) : bs)
37     | n < length cs = Insert (take n cs) : compose' (Insert (drop n cs) : as) bs
38     | otherwise = Insert cs : compose' as bs
39     compose' (Insert cs : as) (Delete n : bs)
40     | n > length cs = compose' as $ Delete (n - length cs) : bs
41     | n < length cs = compose' ((Insert $ drop n cs) : as) bs
42     | otherwise = compose' as bs
43   transform (TextOperation a) (TextOperation b) = pair $ transform' a b where
44     pair (as',bs') = (TextOperation as', TextOperation bs')
45     transform' [] [] = ([],[])
46     transform' (Insert cs : as) bs = case transform' as bs of (as',bs') → (Insert cs : as', Retain (length cs) : bs')
47     transform' as (Insert cs : bs) = case transform' as bs of (as',bs') → (Retain (length cs) : as', Insert cs : bs')
48     transform' (Retain n : as) (Retain n' : bs)
49     | n > n' = case transform' (Retain (n-n') : as) bs of (as',bs') → (Retain n' : as', Retain n' : bs')
50     | n < n' = case transform' as (Retain (n'-n) : bs) of (as',bs') → (Retain n : as', Retain n : bs')
51     | otherwise = case transform' as bs of (as',bs') → (Retain n : as', Retain n' : bs')
52     transform' (Delete n : as) (Delete n' : bs)
53     | n > n' = transform' (Delete (n-n') : as) bs
54     | n < n' = transform' as $ Delete (n'-n) : bs
55     | otherwise = transform' as bs
56     transform' (Retain n : as) (Delete n' : bs)
57     | n > n' = case transform' (Retain (n-n') : as) bs of (as',bs') → (as', Delete n' : bs')
58     | n < n' = case transform' as (Delete (n'-n) : bs) of (as',bs') → (as', Delete n : bs')
59     | otherwise = case transform' as bs of (as',bs') → (as', Delete n : bs')
60     transform' (Delete n : as) (Retain n' : bs)
61     | n > n' = case transform' (Delete (n-n') : as) bs of (as',bs') → (Delete n' : as', bs')
62     | n < n' = case transform' as (Retain (n'-n) : bs) of (as',bs') → (Delete n : as', bs')
63     | otherwise = case transform' as bs of (as',bs') → (Delete n : as', bs')

```

Client

Um leere Retains (durch die neue Darstellung) zu vermeiden, wurden einige zusätzliche Fälle zur Hinzufügfunktion von Zeichen hinzugefügt. Ansonsten funktioniert diese Funktion allerdings analog zur vorherigen Version.

```

11 def insert(c: Char) = {
12   val operation = if (cursor == 0 && content.length == 0) {
13     List(Insert(c.toString()))
14   } else if (cursor == 0) {
15     List(Insert(c.toString()), Retain(content.length))
16   } else if (content.length == 0) {
17     List(Insert(c.toString()))
18   } else if (content.length == cursor) {
19     List(Retain(cursor), Insert(c.toString()))
20   } else {
21     List(Retain(cursor), Insert(c.toString()), Retain(content.length - cursor))
22   }
23   (Option(operation), Editor(TextOperation.applyOp(content, operation), cursor + 1,
24     cursors.map(c => TextOperation.transformCursor(operation, c))))
25 }
26
27 def backspace = if (cursor > 0) {
28   val operation = List(Retain(cursor - 1), Delete(1), Retain(content.length - cursor)).filter(op => op != Retain(0))
29   (Option(operation), Editor(TextOperation.applyOp(content, operation), cursor - 1,
30     cursors.map(c => TextOperation.transformCursor(operation, c))))
31 } else (None, this)
32
33 def moveLeft = {
34   val operation = List(Retain(content.length))
35   (Option(operation), Editor(content, Math.max(0, cursor - 1), cursors))

```

```

36 }
37
38 def moveRight = {
39   val operation = List(Retain(content.length))
40   (Option(operation), Editor(content, Math.min(content.length, cursor + 1), cursors))
41 }

```

Die neue Message *HelloGuys* wird nun verarbeitet und fügt einen neuen Cursor im jeweiligen Client hinzu. Dieser speichert die Cursor der anderen Clients nun in seinem Editor. *RemoteEdits* wurden so angepasst, dass nun der Cursor des Client durch die *RemoteEdit*-Operation transformiert werden muss. Zusätzlich, da nun in einem *RemoteEdit* die Cursor des mit dem Server konsistenten Zustandes mitgesandt werden, müssen diese ebenfalls in einen mit dem jeweiligen Client konsistenten Zustand gebracht werden. Dies bedeutet, die Cursor mit der Operation aus dem pending und dem buffer zu transformieren. Da pending und buffer auch leer sein können, werden diese beiden Optionen abgeprüft.

```

58 val editor = Flow[ClientEvent].scan((ClientState.empty, Option.empty[ClientMessage])) {
59   case ((state, lastMessage), Receive(message)) => message match {
60     case Ack =>
61       val (op, newClient) = state.ot.ack
62       val message = op.map(ClientEdit(newClient.revision, _, 1))
63       (state.copy(ot = newClient), message)
64     case HelloGuys =>
65       val newCursors = state.editor.cursors :+ 0
66       (ClientState(state.ot, Editor(state.editor.content, state.editor.cursor, newCursors)), None)
67     case RemoteEdit(op, cursors) =>
68       val (top, newClient) = state.ot.remoteEdit(op)
69       val newContent = TextOperation.applyOp(state.editor.content, top)
70       val newCursorPos = TextOperation.transformCursor(op, state.editor.cursor)
71       val pendingCursors = state.ot.pending match {
72         case None => cursors
73         case Some(o) => cursors.map(c => TextOperation.transformCursor(o, c))
74       }
75       val newCursors = state.ot.buffer match {
76         case None => cursors
77         case Some(o) => cursors.map(c => TextOperation.transformCursor(o, c))
78       }
79       val newState = state.copy(
80         ot = newClient,
81         editor = Editor(newContent, Math.min(newContent.length, newCursorPos), newCursors)
82       )
83       (newState, None)
84   }
85   case ((state, lastMessage), Keystroke(key)) =>
86     val (op, newEditor) = key match {
87       case KeyValue.Backspace => state.editor.backspace
88       case KeyValue.ArrowLeft => state.editor.moveLeft
89       case KeyValue.ArrowRight => state.editor.moveRight
90       case KeyValue.Enter => state.editor.insert('\n')
91       case other if other.length == 1 => state.editor.insert(other.head)
92       case other => (None, state.editor)
93     }
94     val (newClient, message) =
95       op.fold(
96         (state.ot, Option.empty[ClientMessage])
97       ) { op =>
98         val (syncd, newState) = state.ot.localEdit(op)
99         val msg = if (syncd) Some(ClientEdit(newState.revision, op, newEditor.cursor)) else None
100         (newState, msg)
101       }
102     (ClientState(newClient, newEditor), message)
103   }.recover {
104     case NonFatal(e) =>
105       (ClientState(Client.empty(TextOperation), Editor(e.getMessage, 0, List())), None)
106   }

```

Analog zu dem Server wurden die Funktionen *transformCursor* und *applyOp* auf die neue Datenrepräsentation angepasst.

```

117 def transformCursor(op: Operation, cursor: Int) : Int =
118   TextOperation.split(op, cursor).foldRight(cursor) {
119     (o, acc) => o match {
120       case Retain(n) => acc
121       case Delete(n) => acc - n
122       case Insert(cs) => acc + cs.length
123     }
124   }
125
126 def split(op: Operation, c : Int) : Operation = {
127   op match {
128     case Nil => Nil
129     case (x :: xs) =>
130       if (c <= 0) Nil
131       else x match {
132         case Retain(n) =>
133           if (n <= c) x :: split(xs, c-n)
134           else List(Retain(c))
135         case Delete(n) =>
136           if (n <= c) x :: split(xs, c-n)
137           else List(Delete(c))
138         case Insert(cs) =>
139           if (cs.length <= c) x :: split(xs, c-cs.length)
140           else List(Insert(cs.take(c)))
141       }
142   }
143 }
144
145 def applyOp(doc: Document, op: Operation): Document = (doc, op) match {
146   case ("", Nil) => ""
147   case (d, Retain(0) :: as) => applyOp(d, as)

```

```

147     case (d, Retain(n) :: as) => if d.nonEmpty => d.head + applyOp(d.tail, Retain(n-1) :: as)
148     case (d, Insert(cs) :: as) => cs #+ applyOp(d, as)
149     case (d, Delete(0) :: as) => applyOp(d, as)
150     case (d, Delete(n) :: as) => if d.nonEmpty => applyOp(d.tail, Delete(n-1) :: as)
151 }

```

Da nun zusätzliche Cursor gerendert werden müssen, wurde die *render*-Funktion angepasst, sowie dem eigenen Cursor eine rote Farbe (zur besseren Unterscheidung) gegeben.

```

59 def render(elem: html.Div) = {
60   elem.textContent = content
61   val (w,h) = measure(content.take(cursor))
62   val ownCursor = document.createElement("div").asInstanceOf[html.Div]
63   ownCursor.style.left = w + "px"
64   ownCursor.style.top = h + "px"
65   ownCursor.classList.add("ownCaret")
66   elem.appendChild(ownCursor)
67
68   cursors.foldRight(elem)((c,e) => {
69     val (w,h) = measure(content.take(c))
70     val cursor = document.createElement("div").asInstanceOf[html.Div]
71     cursor.style.left = w + "px"
72     cursor.style.top = h + "px"
73     cursor.classList.add("caret")
74     e.appendChild(cursor)
75     e
76   })
77 }

```

Analog zum Server wurden *transform* und *compose* auch auf dem Client für die neue effizientere Repräsentation der Aktionen angepasst.

```

35 def compose(a: Operation, b: Operation): Operation = (a,b) match {
36   case (Nil, Nil) => Nil
37   case (Delete(n) :: as, bs) => Delete(n) :: compose(as, bs)
38   case (as, Insert(cs) :: bs) =>
39     Insert(cs) :: compose(as, bs)
40   case (Retain(n) :: as, Retain(n2) :: bs) =>
41     if (n > n2) Retain(n2) :: compose(Retain(n-n2) :: as, bs)
42     else if (n < n2) Retain(n) :: compose(as, Retain(n2-n) :: bs)
43     else Retain(n) :: compose(as, bs)
44   case (Retain(n) :: as, Delete(n2) :: bs) =>
45     if (n > n2) Delete(n2) :: compose(Retain(n-n2) :: as, bs)
46     else if (n < n2) Delete(n) :: compose(as, Retain(n2-n) :: bs)
47     else Delete(n) :: compose(as, bs)
48   case (Insert(cs) :: as, Retain(n) :: bs) =>
49     if (n > cs.length) Insert(cs) :: compose(as, Retain(n-cs.length) :: bs)
50     else if (n < cs.length) Insert(cs.take(n)) :: compose(Insert(cs.drop(n)) :: as, bs)
51     else Insert(cs) :: compose(as, bs)
52   case (Insert(cs) :: as, Delete(n) :: bs) =>
53     if (n > cs.length) compose(as, Delete(n-cs.length) :: bs)
54     else if (n < cs.length) compose(Insert(cs.drop(n)) :: as, bs)
55     else compose(as, bs)
56 }
57
58 def transform(a: Operation, b: Operation): (Operation, Operation) = (a,b) match {
59   case (Nil, Nil) => (Nil, Nil)
60   case (Insert(cs) :: as, bs) =>
61     val (as_, bs_) = transform(as, bs)
62     (Insert(cs) :: as_, Retain(cs.length) :: bs_)
63   case (as, Insert(cs) :: bs) =>
64     val (as_, bs_) = transform(as, bs)
65     (Retain(cs.length) :: as_, Insert(cs) :: bs_)
66   case (Retain(n) :: as, Retain(n2) :: bs) =>
67     if (n > n2) {
68       val (as_, bs_) = transform(Retain(n-n2) :: as, bs)
69       (Retain(n2) :: as_, Retain(n2) :: bs_)
70     }
71     else if (n < n2) {
72       val (as_, bs_) = transform(as, Retain(n2-n) :: bs)
73       (Retain(n) :: as_, Retain(n) :: bs_)
74     }
75     else {
76       val (as_, bs_) = transform(as, bs)
77       (Retain(n) :: as_, Retain(n2) :: bs_)
78     }
79   case (Delete(n) :: as, Delete(n2) :: bs) =>
80     if (n > n2) {
81       transform(Delete(n-n2) :: as, bs)
82     }
83     else if (n > n2) {
84       transform(as, Delete(n2-n) :: bs)
85     }
86     else {
87       transform(as, bs)
88     }
89   case (Retain(n) :: as, Delete(n2) :: bs) =>
90     if (n > n2) {
91       val (as_, bs_) = transform(Retain(n-n2) :: as, bs)
92       (as_, Delete(n2) :: bs_)
93     }
94     else if (n < n2) {
95       val (as_, bs_) = transform(as, Delete(n2-n) :: bs)
96       (as_, Delete(n) :: bs_)
97     }
98     else {
99       val (as_, bs_) = transform(as, bs)
100       (as_, Delete(n) :: bs_)
101     }
102   case (Delete(n) :: as, Retain(n2) :: bs) =>
103     if (n > n2) {
104       val (as_, bs_) = transform(Delete(n-n2) :: as, bs)
105       (Delete(n2) :: as_, bs_)

```

```

106     }
107     else if(n < n2) {
108         val (as_, bs_) = transform(as, Retain(n2-n) :: bs)
109         (Delete(n) :: as_, bs_)
110     }
111     else {
112         val (as_, bs_) = transform(as, bs)
113         (Delete(n) :: as_, bs_)
114     }
115 }

```

Eine Reduce-Funktion wurde hinzugefügt, da mehrmalig aufeinanderfolgende *Inserts* zusammengefasst werden können, welche innerhalb von *localEdit* benutzt wird, da dort neue *Inserts* für das gesamte Sheet-System hinzugefügt werden.

```

174 def localEdit(op: ot.Operation): (Boolean, Client[T]) = {
175     val newPending = pending orElse Some(op)
176     val newBuffer = for {
177         pending ← pending
178     } yield buffer.map(ot.compose(_, op)).map(ot.reduce(_)).getOrElse(op)
179     (pending.isEmpty, copy(pending = newPending, buffer = newBuffer))
180 }

```

Tests

Die Cursor sowie die Konsistenz der Cursors sowie des Sheet-Inhaltes auf allen jeweiligen Clients wurde durch Tests mit mehreren (mehr als 3) Clients und gleichzeitiger Bearbeitung des Sheets getestet. Alle Tests wurden hierbei erfüllt.

Dass die effizientere Repräsentation der TextAktionen verwendet wurde sowie das *reduce* funktioniert, wurde mithilfe von prints und logs auf Client- sowie Serverseite getestet und die Tests wurden bestanden.