# Position that i will fix

## First change

```
interface Props extends BoxProps {}
```

> I don see BoxProps so i will change it

```
interface Props {
  children?: ReactNode;
}
```

## Second change

```
const sortedBalances = useMemo(() => {
  return balances
    .filter((balance: WalletBalance) => {
      const balancePriority = getPriority(balance.blockchain);
      if (lhsPriority > -99) {
        if (balance.amount <= 0) {
          return true;
        }
      }
      return false;
    })
    .sort((lhs: WalletBalance, rhs: WalletBalance) => {
      const leftPriority = getPriority(lhs.blockchain);
      const rightPriority = getPriority(rhs.blockchain);
      if (leftPriority > rightPriority) {
        return -1;
      } else if (rightPriority > leftPriority) {
        return 1;
      }
    });
}, [balances, prices]);
```

> **In the filter function, you use lhsPriority instead of BalancePriority, this is wrong**

> **Reformat result of filter**

> **Use return rightPriority - leftPriority to sort by descending priority.**

```
const sortedBalances = useMemo(() => {
  return balances
    .filter((balance: WalletBalance) => {
      const balancePriority = getPriority(balance.blockchain);
      return balancePriority > -99 && balance.amount <= 0;
    })
    .sort((lhs: WalletBalance, rhs: WalletBalance) => {
      const leftPriority = getPriority(lhs.blockchain);
      const rightPriority = getPriority(rhs.blockchain);
      return rightPriority - leftPriority;
    });
}, [balances, prices]);
```

# Third change

```
const usdValue = prices[balance.currency] * balance.amount;
```

> **I will check the validity of prices at here**

```
const usdValue = prices?[balance.currency] * balance.amount;
```

# Fourth change

```
const getPriority = (blockchain: any): number => {
  switch (blockchain) {
    case "Osmosis":
      return 100;
    case "Ethereum":
      return 50;
    case "Arbitrum":
      return 30;
    case "Zilliqa":
      return 20;
    case "Neo":
      return 20;
    default:
      return -99;
  }
};
```

> **Add type for blockchain**

```
type BlockChange = "Osmosis" | "Ethereum" | "Arbitrum" | "Zilliqa" | "Neo";

const getPriority = (blockchain: BlockChange): number => {
  switch (blockchain) {
    case "Osmosis":
      return 100;
    case "Ethereum":
      return 50;
    case "Arbitrum":
      return 30;
    case "Zilliqa":
      return 20;
    case "Neo":
      return 20;
    default:
      return -99;
  }
};
```

# Final all code after fix

```tsx
interface WalletBalance {
  currency: string;
  amount: number;
}
interface FormattedWalletBalance {
  currency: string;
  amount: number;
  formatted: string;
}
type BlockChange = "Osmosis" | "Ethereum" | "Arbitrum" | "Zilliqa" | "Neo";

interface Props {
  children?: ReactNode;
}
const WalletPage: React.FC<Props> = (props: Props) => {
  const { children, ...rest } = props;
  const balances = useWalletBalances();
  const prices = usePrices();

  const getPriority = (blockchain: BlockChange): number => {
    switch (blockchain) {
      case "Osmosis":
        return 100;
      case "Ethereum":
        return 50;
      case "Arbitrum":
        return 30;
      case "Zilliqa":
        return 20;
      case "Neo":
        return 20;
      default:
        return -99;
    }
  };

 const sortedBalances = useMemo(() => {
    return balances.filter((balance: WalletBalance) => {
        const balancePriority = getPriority(balance.blockchain);
        return balancePriority > -99 && balance.amount <= 0;
```

```
    }).sort((lhs: WalletBalance, rhs: WalletBalance) => {
        const leftPriority = getPriority(lhs.blockchain);
        const rightPriority = getPriority(rhs.blockchain);
        return rightPriority - leftPriority;
    });
}, [balances, prices]);

  const formattedBalances = sortedBalances.map((balance: WalletBalance) => {
    return {
      ...balance,
      formatted: balance.amount.toFixed(),
    };
  });

  const rows = sortedBalances.map(
    (balance: FormattedWalletBalance, index: number) => {
      const usdValue = prices?[balance.currency] * balance.amount;
      return (
        <WalletRow
          className={classes.row}
          key={index}
          amount={balance.amount}
          usdValue={usdValue}
          formattedAmount={balance.formatted}
        />
      );
    }
  );

  return <div {...rest}>{rows}</div>;
};
```