Dokumentobjectmodell

Was ist das "Document Object Model", oder das DOM?
Das DOM ist der Data Vertretung von Objekte, dass beinhaltet das Aufbauen und der Content ins Web.

> Mehr Informationen: https://www.w3.org/TR/WD-DOM/introduction.html

Durchsführung

Für Rust-Lang:

```rust
use std::{
    borrow::BorrowMut,
    cell::RefCell,
    collections::HashMap,
    rc::{Rc, Weak},
};

use crate::elements::ElementData;

pub type DictDOMWeakRef = Weak<RefCell<DictDOMItem>>;
pub type DictDOMRef = Rc<RefCell<DictDOMItem>>;

#[derive(Debug, Clone, Default)]
pub struct DictDOMItem {
    pub object: ElementData,
    pub data: String,
    pub parent: Option<DictDOMWeakRef>,
    pub children: Vec<DictDOMRef>,
    pub id: Option<String>,
}
```

```
impl DictDOMItem {
    pub fn add_child(&mut self, child: DictDOMRef) {
        self.children.push(child);
    }
    pub fn get_parent(&self) -> Option<DictDOMRef> {
        match &self.parent {
            Some(x) => Weak::upgrade(&x),
            None => None,
        }
    }
    pub fn query_child(&self, query: fn(&&DictDOMItem)
-> bool) -> Vec<DictDOMItem> {
        /* SEE NOTE ../notes/dom.md#Query Child */
        let mut results: Vec<DictDOMItem> =
Vec::new();
        if query(&self) {
            results.push(self.clone());
        }
        for i in &self.children {

results.extend(i.as_ref().borrow().query_child(query))
;
        }
        results
    }
    pub fn get_object(&self) -> &ElementData {
        &self.object
    }
    pub fn get_children(&self) -> &Vec<DictDOMRef> {
        &self.children
    }
    pub fn get_id(&self) -> String {
        match self.id.clone() {
            Some(x) => x,
            None => String::from(""),
        }
    }
    pub fn get_data(&self) -> &String {
        &self.data
    }
    pub fn get_data_owned(&self) -> String {
        self.data.clone()
    }
    pub fn set_object(&mut self, object: ElementData)
{
        self.object = object;
    }
    pub fn set_id(&mut self, id: String) {
        self.id = Some(id);
    }
    pub fn set_data(&mut self, data: String) {
        self.data = data;
    }
}

pub struct DictDomBuilder(DictDOMItem);

impl DictDomBuilder {
    pub fn create() -> Self {
        Self(DictDOMItem::default())
    }
    pub fn object(self, object: ElementData) -> Self {
        Self(DictDOMItem { object, ..self.0 })
    }
    pub fn parent(self, parent: &DictDOMRef) -> Self {
        Self(DictDOMItem {
            parent: Some(Rc::downgrade(parent)),
            ..self.0
        })
    }
    pub fn children(self, children: Vec<DictDOMRef>) -
> Self {
        Self(DictDOMItem { children, ..self.0 })
    }
    pub fn id(self, id: String) -> Self {
        Self(DictDOMItem {
            id: Some(id),
            ..self.0
        })
    }
    pub fn data(self, data: String) -> Self {
        Self(DictDOMItem { data, ..self.0 })
    }
    pub fn build(self) -> DictDOMRef {
        let item = Rc::new(RefCell::new(self.0));
        match &item.as_ref().borrow_mut().parent {
            Some(x) => {
                let mut parent =
Weak::upgrade(&x).expect("Should be a parent!");

parent.as_ref().borrow_mut().add_child(Rc::clone(&item
));
            }
            None => {}
        }
        item
    }
}
```