13.12.2019 // Protokoll Jit.si-Konferenz 09.12.19

- Protokoll Jit.si-Konferenz 09.12.19
  - Teilnehmer
  - “Morphologischer Kasten”
  - Fragen die noch zu klären sind…
    - Kuratiertes Framework vs. Erweiterbarer Monolith
    - Fabian:
    - Was soll beim 36C3 geschehen?
    - Geld beim VoW?!
  - Monolith / Modulsystem / FrameWork??
- nächster Call

Teilnehmer

- Micha (Kiel / WSK)
- Fabi (HRW / Düsseldorf / FabAccess)
- Dequbed (Berlin / BF2H)
- Thee (BHT / Bachelorarbeit TI)
- Marcus Drobisch (Dresden / RoseGuard)
- Kevin (Dresden / RoseGuard)
- Maximilian Voigt (Cottbus / VOW)
- Tasso / Knurps (Berlin / BHT)

“Morphologischer Kasten”

[OAS] [RoseGuarden] [BF2H]

- FabAccess ist Semesterprojekt-Arbeit (18SWS) … und noch zusätzlich durch VOW gepusht. Bis März

MLP / MAP: Sinnvoll … nur, vermeiden dass das Skateboard mit dem Endprodukt vermischt wird & den ersten Eindruck “versaut”.

Entscheidungen: Konsens / Systemisches Konsensionieren

Programmiersprache:

☑ Wollen wir das jetzt entscheiden?

- Frontend ist nicht zwingend Backend.

<Fabian> Klare Empfehlung, Diskussion über Technologie relativ spät. Was sind die UseCases? … Was kann ich aus den UseCases ableiten, sinnvolle Technologien dazu zu finden.</Fabian>

<Marcus>+1 technische Features / Technologien ja … Programmiersprache als Basic früher.</Marcus>
<M V> Vorstellung der bisherigen Systeme … was kann man dort herausziehen? - ergibt sich daraus schon etwas?</M V>

<Fabi>Hilft Sprachwahl auf Meta-Ebene?</Fabi>

[…] Sprachdiskussion.

- funktionale / typstarke Programmiersprachen: Stabiler. Vermeiden Bugs “aus Versehen” einzubauen. Compiler verbietet mehr Fehler.
- Ruby / Python / … breitere Basis an potentiellen Programmierern / Community.
- Sicherheit herstellen durch Testing.
- Sicherheit herstellen durch typisierte Sprache & Testing.
- ☐Rust kann’s / würde es gerne lernen / kann’s nicht / kann’s nicht / kann’s nicht / 3x gerne lernen / 1x wofür? / community wird beschnitten / schnell lernbar (viel Doku) /
- ☐Python halbwegs / kann’s, aber nicht genug für Infrastruktur-Software / nutze es täglich um Messdaten auszuwerten, mittelgut / geht bisschen Datenmanipulation / gut dabei / relativ gerne // ja, gerne / wäre bereit zu lernen / hätte bock, mehr zu machen / kein Bedarf // Doku gibt’s, schnell lernbar, größere Community als Rust, übel viele Libraries. Python2 … ist tot. Python 3.6 hat typechecks, aber so … hmm.
- ☐Java / kann’s ein bisschen, nicht genug für Infrastruktur / kann’s ein bisschen, auch Infrastruktur damit gemacht. / hab’s ein bisschen, hauptsächlich App-Entwicklung / mal 1 Semester mit AndroidStudio / paar Kurse gehabt // nicht wahnsinnig Bock drauf, würd’s aber machen +1 / ungern --> Kotlin +2 // 9 ist in 2000ern angekommen

- ☑Kotlin / kann … keiner? / 2x: lieber bei Java bleiben als Kotlin zu lernen / gerne lernen +1 / Java ohne Java-Fehler (NullPointerException / …) schwer zu lernen.
- ☑Scala IDE aufwärmen / Programmieren lernen / Programmieren lernen / Programmieren lernen / Programmieren lernen // kann / will keiner
- ☐TypeScript kann ich gut / relativ viel JavaScript, bisschen TypeScript und node.js TypeScript: ein wenig, steile Lernkurve / noch nie verwendet, JavaScript doch öfter, leicht zu lernen / äääh, nein. / nicht so viel damit gemacht // überzeugt mich / gerne lernen // kommt aus dem Node-Umfeld, typisiert, strict-mode verhindert viel Unfug, viele schlechte, aber auch viele gute Bibliotheken, große Community, geringe Einstiegshürden, ProtokollImplementation … will man wohl nicht machen, Bibliotheken (z.B. AMQP) sind aber verwendbar. (Viele) Backend-Tasks müssten abgedeckt sein.

Fragen:

- können wir
- stabil
- wie schnell kann man’s lernen?
- wie viel muss man selber machen?

Bewertung (1: Am liebsten 4: Am wenigst liebsten X: VETO):

- dequbed (10h - 20h Back):
  - Rust: 1
  - TypeScript: 2
  - Java: 3
  - Python: 4
- mdrobisch (langfristig viel):
  - Python: 1
  - TypeScript: 2
  - Rust: 3
  - Java: 4
- fabi (15 - 20h Front&Back):
  - Python: 2
  - TypeScript: 1
  - Rust: 4
  - Java: 3
- Thee:
  - Python: 1
  - TypeScript: 3
  - Rust: 2
  - Java: 4

- SUM:
  - Rust: 1 + 3 + 4 + 2 = 10
  - Python: 4 + 1 + 2 + 1 = 8
  - TypeScript: 2 + 2 + 1 + 3 = 8
  - Java: 3 + 4 + 3 + 4 = 14

Frontend: Kevin / Marcus / Fabian /
Backend: Dequbed / Marcus / Fabian / Thee
Firmware Clients: Tasso / Dequbed / Thee

Ranking: Beliebteste Sprache? … Sprache die am häufigsten verwendet wird? … Was kann wer aktuell? … was machen viele? … was machen “die Guten”?

Kotlin - noch nicht so gewachsen wie Java
Java 9 - kaum Dokumentation, aber gewachsen.
Java 5 - viel Dokumentation aber nicht 2020.
Python Umstieg 2 zu 3 … relativ unproblematisch.
Python: Exceptions, daher potentiell instabil. Evtl. abfangbar mit Lint / CI
TypeScript … ist Javascript mit strikter Typisierung --> fängt viele Fehlermöglichkeiten ab.
TypeScript != Node.js
Node kann threaden, erreicht gute Performance, JavaScript ist event-fokussiert, singlethreaded. Node kann zwar threaded, man arbeitet aber ähnlich wie im Browser.
Promises … gibt’s in TypeScript. Alles aus JavaScript + TypSystem. Java: geht, Python: geht (twisted?) … Rust: Standard-Feature

Fragen die noch zu klären sind…

Kuratiertes Framework vs. Erweiterbarer Monolith

Fabian:

- Paper ist zur SensorNets Link angenommen. Kann in der Endversion auf OAS erweitert werden und Öffentlichkeit für unser Projekt zu generieren.
- Alex Roussolet? … europäische Ebene aktivieren / Erasmus für MakerSpaces / FabLabs (Belgien, Frankreich, …)

Was soll beim 36C3 geschehen?

- Technische Fragen, die bis da hin vermutlich mehr als weniger werden “live und in Farbe” … klären!

- Nur am Rande … von 3 Core-Leuten sind 1 da!
- Statt dessen den Workshop-Slot nutzen, um Feedback aus der 36C3-Community abholen. Sicht von außen auf Teile unserer bisherigen Diskussionspunkte.
- UseCases sammeln.

Geld beim VoW?!

- Sozialfonds ist gut.
- Hardware auch.

Monolith / Modulsystem / FrameWork??

Monolithic vs. Modular (JavaScript/Node.js)

Simon Brown - Modular Monoliths

nächster Call

Doodle - lockerer, offener Termin noch vor Weihnachten? … ansonsten intensiver im neuen Jahr.

Version #4
Erstellt: 2024-10-13 00:57:14 CEST von Mario Voigt (Stadtfabrikanten e.V.)
Zuletzt aktualisiert: 2025-02-25 21:22:13 CET von Mario Voigt (Stadtfabrikanten e.V.)