Software Architecture
MSc (Spring 2025)

Course code:

KSSOARC2KU

Group:

The Architects

Members:

Jacob Grum,

Rasmus Ole Routh Herskind &

Thor Liam Møller Clausen

Table of Contents

0 How to read this report

For the shared part:

The most important parts are highlighted with bold. If the reader only need a concise understanding you can stick to these parts.

The shared report covers section headers numbered 1-8

For the individual parts:

- Jacob (jacg): Spans section 9 to 13. Important parts have been highlighted with bold.
- Thor (tcla): Start at page 105 and ends at page 116. Important parts have not been highlighted with bold.
- Rasmus (rher): Start at page 117 and ends at page 125. Important parts have not been highlighted with bold.

1 System description

1.1 Overall functionality

Imagine the following; Kevin wants to bet on Holger winning his tennis match. He checks his usual site, which gives odds 1.8. But his mate next to him scuffs at him "*you should check out my site, it gives odds 1.9*". Kevin quickly realizes that he is not loyal to his site, he is mostly interested in placing on any site with the best odds available - Introducing **BestBets**

BestBets is an **online betting odds aggregator** that provides users with a **unified interface** to find and place bets at the best available odds **across multiple bookmakers**. It eliminates the need for users to manually check different betting sites, by collecting odds data from various bookmakers in real time and allowing bets to be placed through one central platform.

1.1.1 Key features of BestBets

1. **Shows the best odds** available **from any bookmaker** for any match. (Fig 1)

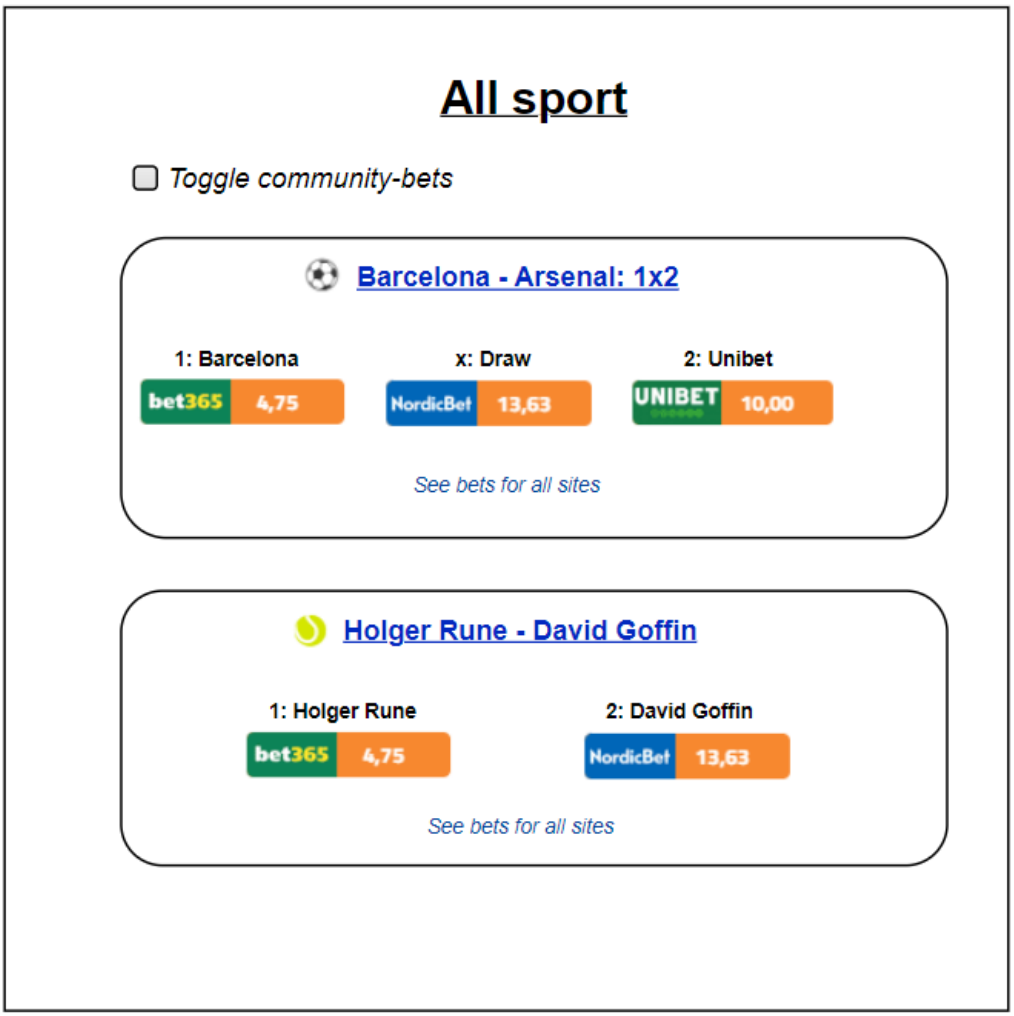


Figure 1: Simple example match overview, showing the current best odds from various bookmakers.

2. Plenty of stock-like features! **Trigger orders,** when a certain game reaches certain odds. (Fig 2)

Figure 2: Simple example of placing a trigger order on a specific match

3. Much like trading stocks on a central exchange ("Børsen") **you only need one central balance to wager with any bookmaker**. For example, a 200 kr deposit on BestBets lets you place up to 200 kr on any bookmaker—no separate accounts or individual deposits required (Fig 3)

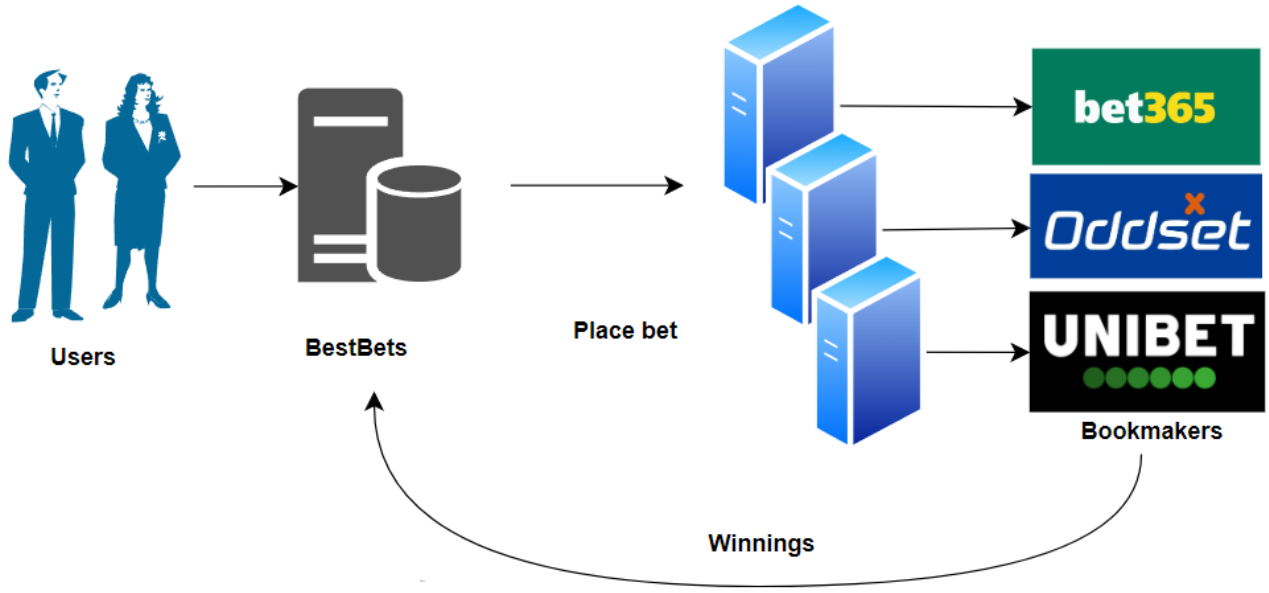


Figure 3: Illustration of how all money is centralized on BestBets both when placing bets and winning money

4. Think the best odds are bad? **Open your own position**, and **act like the bookmaker**! (Fig 4). *Side-note: We ended up not going in-depth with this feature.*

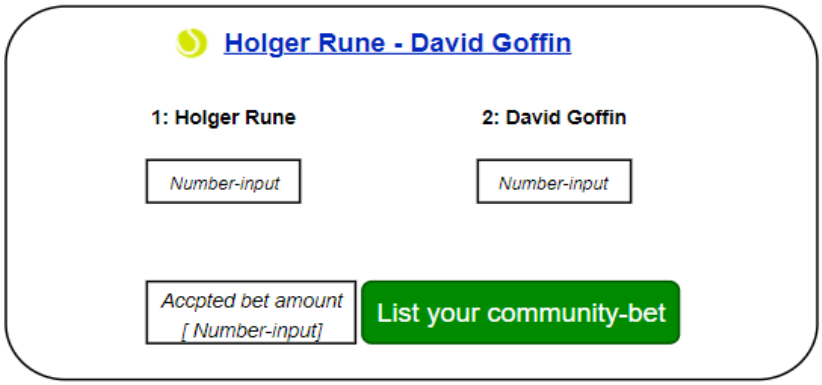


Figure 4: Simple example of how one can open their own odds on a specific match, and as such acts as their own bookmaker

5. **Quickly register for any bookmaker** through a saved formula, to get **consistent information across multiple sites**, and to always be ready to place on the best odds possible. (Fig 5)

Figure 5: Simple example of how a user can quickly register with consistent information across various bookmakers

6. Get an **instant overview** of **current bets**, **accounts** and **bonuses**! (Fig 6)

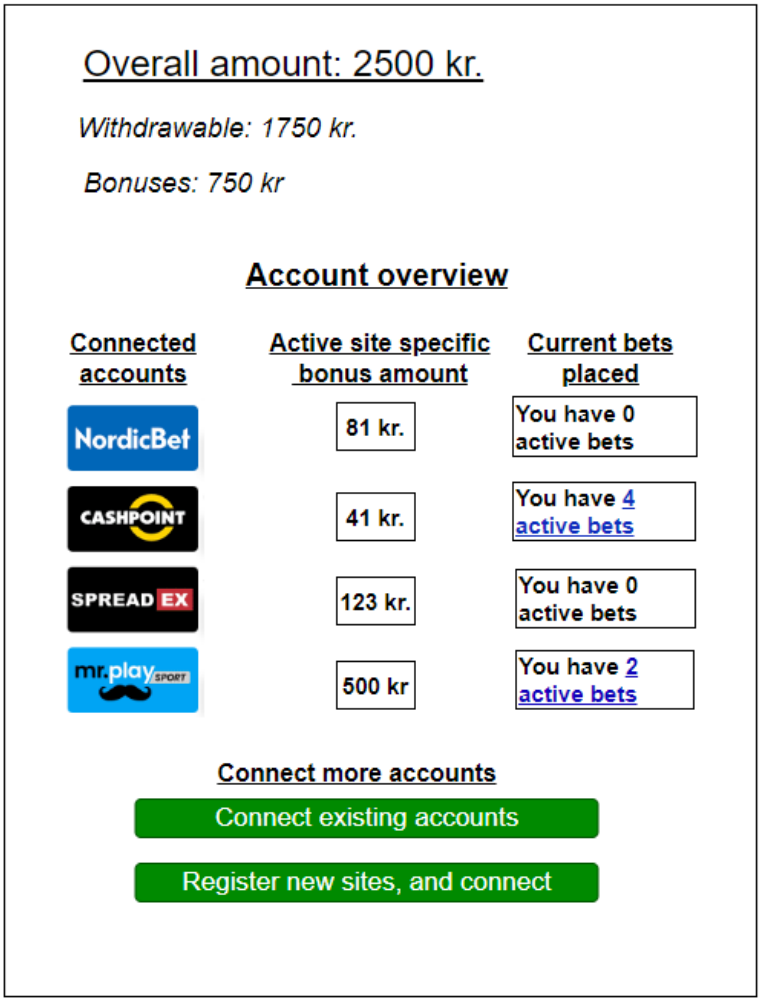


Figure 6: Simple account overview example, showing the most important betting information, such as bonuses and active bets

7. **Various settings** to ensure only your favorite sports are shown to ease navigation and usage. (Fig 7)

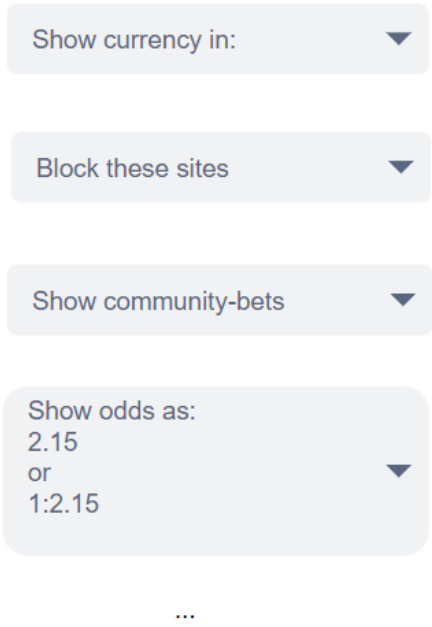


Figure 7: Simple example of various common settings found on betting sites

1.2 Context diagrams

1.2.1 Informal diagram

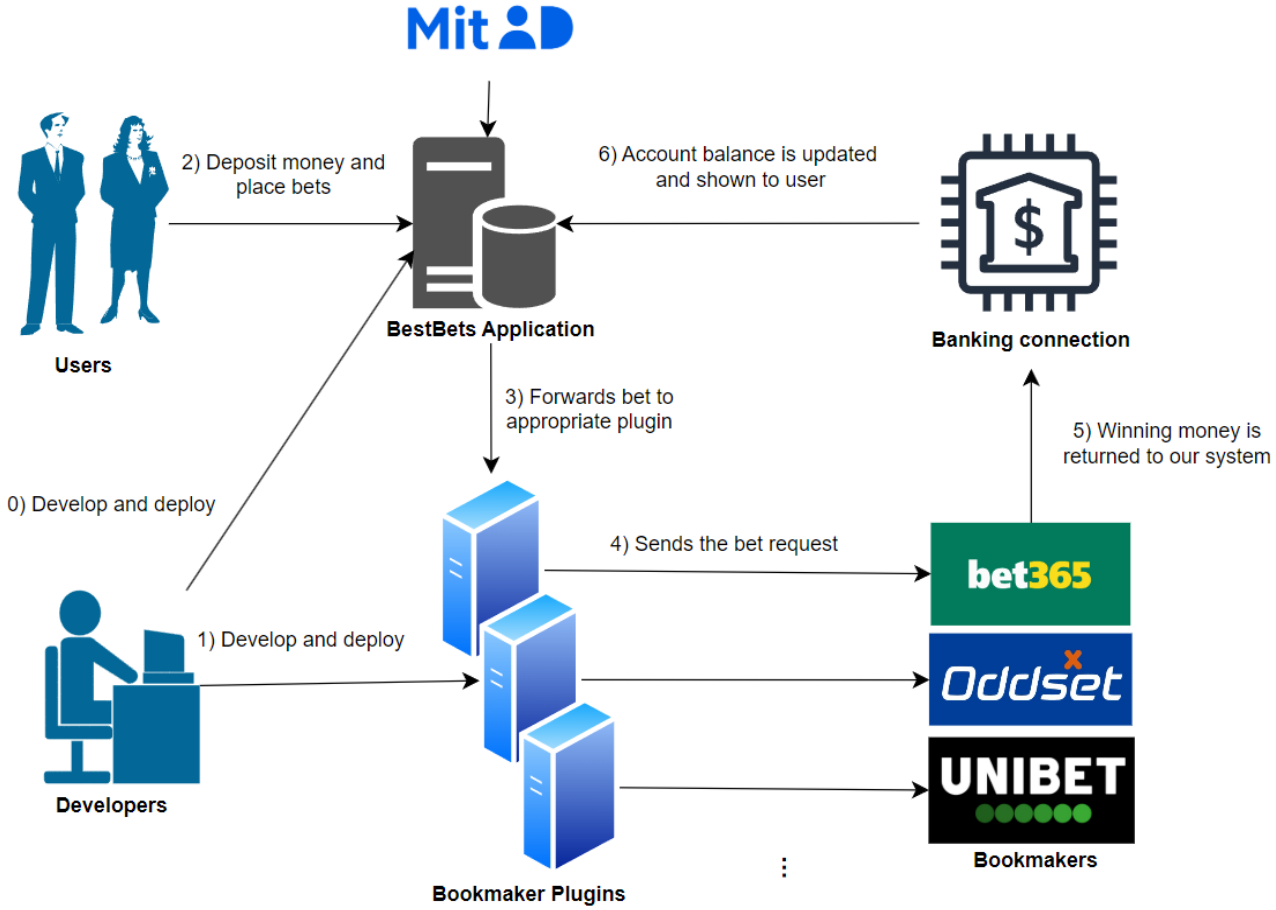


Figure 8: Informal diagram of the BestBets architecture.

As seen in Figure 8, **users only interact with our site. We forward the correct placements/winnings**, so they are always **centralized on BestBets**. They only time they see another bookmaker is during registration.

1.2.2 Formal diagram

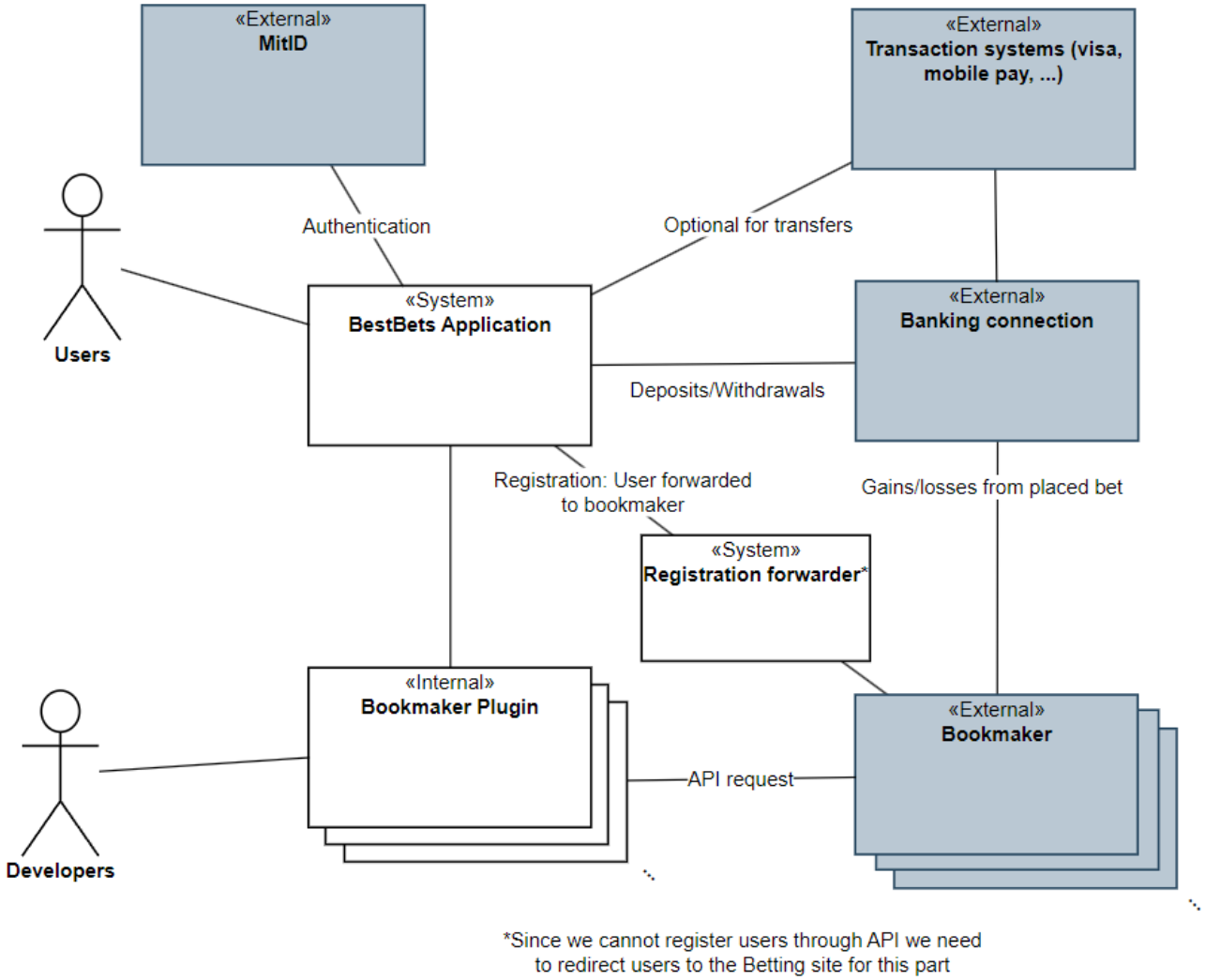


Figure 9: Formal context diagram of BestBets that details the actors (developers, users, external bookmakers) and interfaces they use

As we are a gambling platform we are subjected to various laws from **Spillemyndighederne** (Spillemyndigheden, 2025). Therefore, we are **required to involve MitId** during authentication.

It is also quite common for these sites to offer a **wide range of payment options**. So, we utilize both a banking connection for account transfers, as well as additional payment providers such as mobilepay, visa and mastercard. Regardless of payment method, the money is kept in our external banking connection.

2 Architectural analysis

2.1 Identification of Architectural Drivers

The numbers in parenthesis are the number of points given in our stakeholder role play.

Performance Requirements:

- **Low latency (14)**
  The system shall fetch and display betting odds from multiple bookmakers in real-time or near-real-time. Delays in page loads or data updates must be minimized to prevent users from missing favorable odds.
- **Efficient data processing** (**14)**
  The system must process large volumes of betting data rapidly (especially during peak activity such as major finals) ensuring robust performance under heavy load.

Modifiability Requirement:

- **Handle new bookmakers (13)**
  The system should be modifiable, such that we can easily add or remove bookmakers with minimal impact on the overall system, as that operation will often be needed (Spillemyndigheden, 2025).

Safety Requirement:

- **Safely handle odds changes (12)**
  The system shall ensure safety during the bet placement and acceptance process. If betting odds change between the time a bet is placed and accepted, the system must reject the altered odds and prompt the customer to re-confirm their bet under the new conditions. This process is later referred to as a retry.

Availability Requirement:

- **Available during high demand (12)**
  The system must remain available even during periods of high user demand, e.g. during high-traffic sporting events, to ensure continuous access to betting.

Usability Requirements:

- **Intuitive navigation (11)**
  Users shall be able to quickly and easily find the best odds for a match.
- **Support for filtering (11)**
  Users shall be able to filter for e.g. tennis matches with Holger Rune as participant that ends within 12 hours.
- **Consistent design (11)**
  Bets placed on bookmaker X must have same mechanics and appearance as on bookmaker Y.

- **Responsive design (10)**
  The interface must adapt seamlessly to various devices (mobile, tablet, desktop) to accommodate users checking odds on the go.

Security Requirement:

- **The system must ensure robust security (9)**
  User data and communications must be safeguarded using secure handling mechanisms. Given that customers place real money on bets, the system must ensure that all financial and personal data are protected from unauthorized access.

In summary, the most important ones addressed for the rest of the report:

1. **Performance** (14)
   - Low latency: Provide fresh odds.
   - Efficient Data Processing: During many bets placed during big event
2. **Modifiability** (13)
   - In case many new bookmakers open or existing ones close, it must be handled without too much trouble.
3. **Safety** (12)
   - With money being handled, the user must not be prone to make mistakes.

2.2 Stakeholder roles

Often directly in touch with the system

- Bookmakers
- Developers/Architects
- Users

Various gambling related governmental instances

- Danish gambling authorities: https://www.spillemyndigheden.dk/gaming-og-gambling
- Gambling addiction support associations : https://www.stopspillet.dk/

Various partners and affiliates

- Sport-event industry (Arenas, leagues, etc).
- Sport clubs
- Sports news (magazines, blogs, etc.)
- Sports apparel and merchandise
- Sport streaming & channels

2.3 Quality Attribute Scenarios

2.3.1 Performance

Refined Scenario 1: Real-Time Betting Odds Update (Fig 10)

Scenario: The system updates betting odds in real-time during a high-profile sports event.
Relevant quality: Performance
Stimulus source: External bookmakers.
Stimulus: A sudden surge in updated odds from multiple bookmakers as a major match reaches a critical point.
Environment: System is running high traffic during live, high-stakes sports events.
Artifact: The module(s) that fetches live data from betting sites and updates our site.
Response: Fetch and process incoming data rapidly and update the user interface with minimal delay.
Response measure: Updated odds are displayed within 500 milliseconds of detected change, sustaining throughput of up to 100.000 updates per minute without performance degradation.

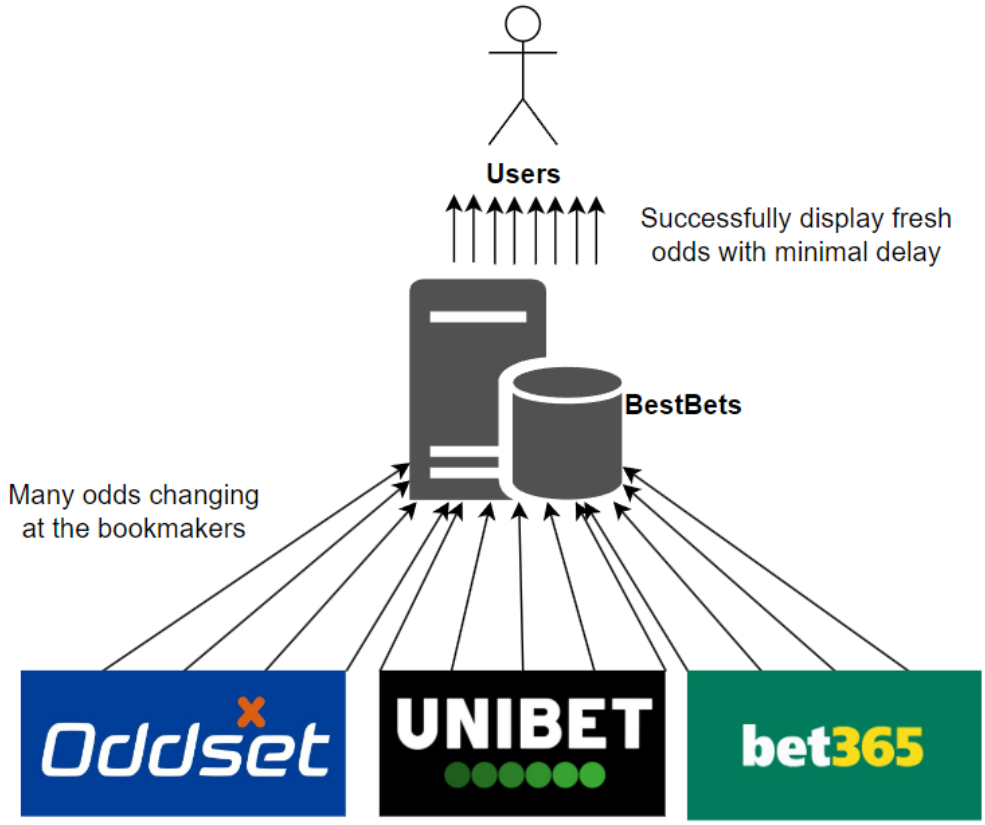


Figure 10: Informal diagram of performance scenario: Real-Time Betting Odds Update.

<u>Refined Scenario 2: Massive user surge during the Champions League final</u> (Fig 11)

Scenario: The system responds to a lot of users during Champions League final with little delay.
Relevant quality: Performance/Scalability
Stimulus source: A lot of end users (100.000) logging in at the same time.
Stimulus: Number of users logged in increases from 1000 to 100.000 in the span of 10 minutes.
Environment: System running normally.
Artifact: Most of the modules that takes user input.
Response: The system continues to serve updated odds without errors or significant delays
Response measure: Pages load within 2 seconds for 95% of requests. Odds update within 3 seconds of bookmaker feed changes.

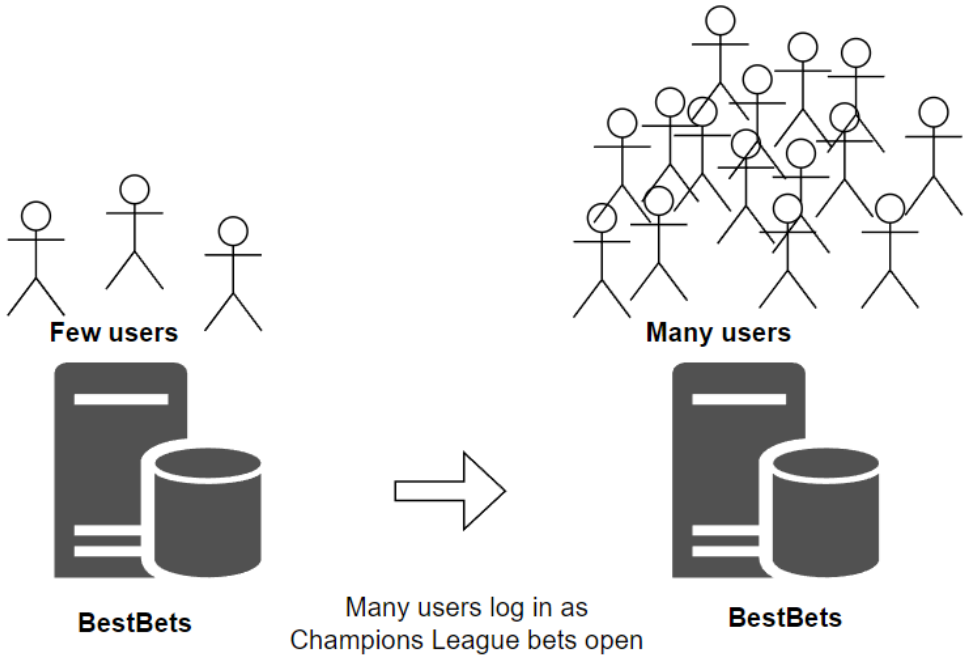


Figure 11: Informal diagram of performance scenario 2.

Refined Scenario 3: Massive surge of placed bets during a major sports match

Scenario: The systems handles a lot of bets placed within a short time span with little delay.
Relevant quality: Performance
Stimulus source: A lot of end users (100.000) placing bets.
Stimulus: 100.000 bets being placed per 5 minutes.
Environment: The system running at high capacity with many users logged in.
Artifact: The module(s) handling the forwarding of bets to the bookmakers, and module(s) receiving confirmation from them.
Response: The system successfully forwards the bet placements without creating bottlenecks.
Response measure: With 100.000 bets per 5 minutes the average latency should be below 1 second from user confirms in our system to the bet being successfully forwarded to the respective bookmaker.

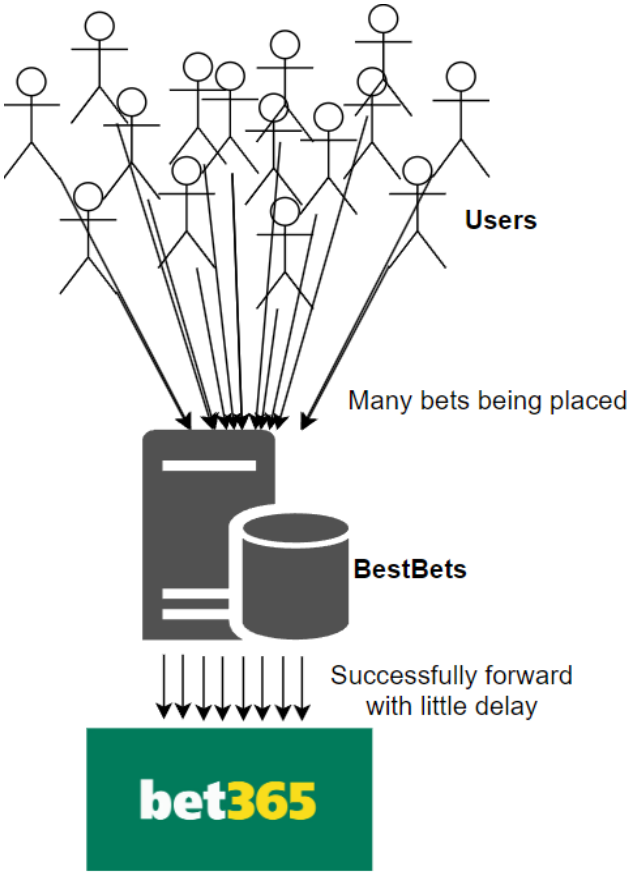


Figure 12: Informal diagram of performance scenario 3.

2.3.2 Availability

Refined Scenario 1: Continuous Availability Under Traffic Spike

Scenario: The platform remains fully available during an unexpected surge in user traffic.
Relevant quality: Availability
Stimulus source: End users accessing the site during a major sports event.
Stimulus: 100.000 users simultaneously accessing and interacting with the platform.
Environment: System running on high capacity from the many logged in users.
Artifact: The modules handling user activity (front end, server, load balancers, auto-scaling infrastructure).
Response: The system dynamically scales resources and distributes the load to maintain service availability.
Response measure: Achieves 99.9% up-time during peak load with 95% of requests served in under 500 milliseconds.

Refined Scenario 2: Site remains available during bookmaker outages

Scenario: The system handles the failure of one or more bookmakers without affecting overall availability.
Relevant quality: Availability
Stimulus source: Bookmakers failing to respond to requests (ping).
Stimulus: Ping requests without a response within 2 seconds from one or more bookmakers.
Environment: System running normally with logged in end users.
Artifact: Modules communicating with bookmakers (checking for activity, placing bets etc).
Response: The system continues to display odds from other bookmakers and clearly labels outed bookmaker as “unavailable”.
Response measure: The system has a 99.99% up-time with up to 100% of bookmakers unavailable.

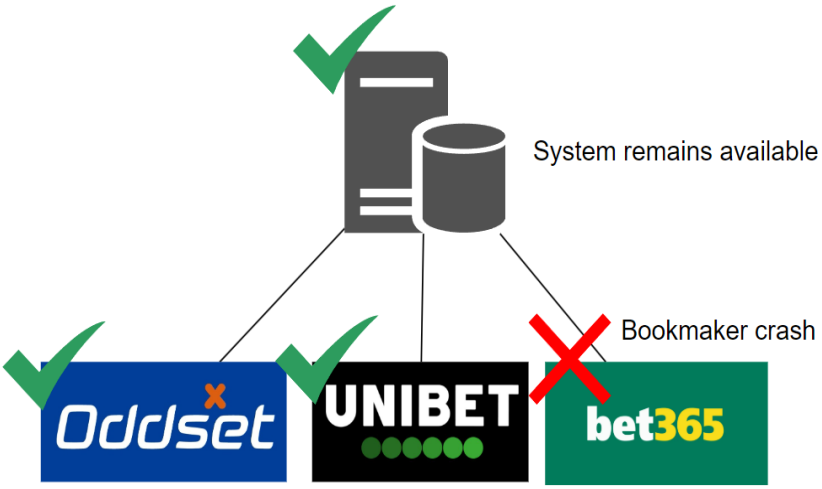
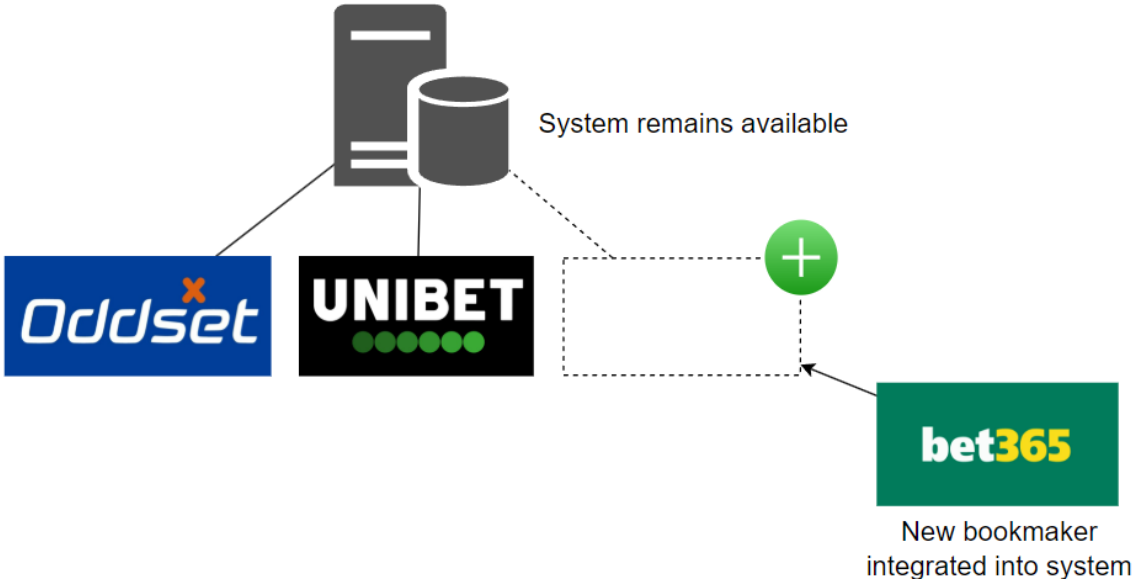


Figure 13: Informal diagram of availability scenario 2.

Refined Scenario 3: The integration of a new bookmaker does not affect availability

Scenario: A new bookmaker has been integrated in the system and gets deployed without affecting active users.
Relevant quality: Availability
Stimulus source: The developers deploying a new bookmaker plugin to the system.
Stimulus: Initiation of the deployment of the new bookmaker plugin.
Environment: System running normally with logged in end users.
Artifact: The modules interacting with the bookmaker plugins.
Response: The system switches seamlessly to the updated version without service disruption.
Response measure: Zero downtime is observed; the switchover is completed seamlessly with monitoring confirming uninterrupted service.



2.3.3 Usability

Refined Scenario 1: Consistent Navigation Across Devices

Scenario: The system provides a uniform and consistent user experience across various devices.
Relevant quality: Usability
Stimulus source: End users accessing the system on desktops, tablets, and mobile devices.
Stimulus: A user transitions between devices and expects the interface to behave consistently.
Environment: System running normally.
Artifact: User interface.
Response: The user interface maintains consistent layout, design, and navigation regardless of the device.
Response measure: Less than 15% difference in the speed of user interaction across device.

Refined Scenario 2: Intuitive Filtering and Search

Scenario: Users quickly locate and filter betting events to find their preferred matches.
Relevant quality: Usability
Stimulus source: End users utilizing search and filtering tools.
Stimulus: A user applies filters (e.g., for tennis matches or events ending within 12 hours) to quickly locate relevant betting options.
Environment: System running normally with search/filter options implemented.
Artifact (if known): UI components and logical components managing search and filtering.
Response: The system delivers relevant results promptly, making it easy for users to narrow down options.
Response measure: At least 90% of search queries return results within 300 milliseconds; user feedback on the filtering process is overwhelmingly positive (95% satisfaction).

Refined Scenario 3: Streamlined, Consistent Bet Placement Process

Scenario: The bet placement workflow is intuitive and consistent across different sections of the site.
Relevant quality: Usability
Stimulus source: A user wanting to place a bet on several different betting sites using our system.
Stimulus: The user navigating through the system to place their bets.
Environment: System running normally.
Artifact (if known): The UI allowing for bet placement.
Response: The user successfully placing their bets without much delay in between bets.
Response measure: 90% of users successfully placing bets in less than 2 minutes on average and in case of multiple bets, at most 1 minute between bets on average.

2.3.4 Modifiability

Refined Scenario 1: Adding a New Bookmaker

Scenario: The system integrates a new bookmaker with minimal disruption.
Relevant quality: Modifiability
Stimulus source: A business decision to expand service offerings.
Stimulus: Announcement of a new bookmaker entering the market.
Environment: Development and production environments.
Artifact: Bookmaker plugins.
Response: Development team implements the new bookmaker interface, tests it, and deploys it with minimal disruption, and minimal changes to the existing code base.
Response measure: Completed integration within 5 developer days. No existing functionality broken (automated tests pass). Less than 5% of the existing code base changed.

Refined Scenario 2: Removing an Existing Bookmaker

Scenario: The system gracefully removes support for a bookmaker that is ceasing operations.
Relevant quality: Modifiability
Stimulus source: External bookmaker shutting down.
Stimulus: An existing bookmaker has been decided to be removed from the system.
Environment: The system running with multiple integrated bookmakers.
Artifact: Bookmaker plugins.
Response: The bookmaker plugin is isolated for safe removal.
Response measure: Plugin removal is executed in under 24 hours with zero system downtime.

Refined Scenario 3: Adapting to Changing Bookmaker APIs

Scenario: The system is modified to handle bookmaker's changed API with minimal impact on overall functionality.
Relevant quality: Modifiability
Stimulus source: External bookmaker API.
Stimulus: The bookmaker releases a new version of its API.
Environment: System running with multiple integrated bookmakers.
Artifact: Bookmaker plugin.
Response: The plugin is updated to accommodate the API changes while keeping the core system unaffected.
Response measure: The required modifications are completed and fully tested within 48 hours with only localized changes in the plugin code.

2.3.5 Safety

Refined Scenario 1: Acceptance of changed odds

Scenario: There is a difference in the odds between the time the user enters its desired bet, and when the bet is being confirmed at the bookmaker.
Relevant quality: Safety
Stimulus source: Bookmaker.
Stimulus: A user enters a bet and, immediately afterward, the betting odds change before confirmation.
Environment: System is running normally.
Artifact: Module(s) handling placement of bets.
Response: The initial desired bet is rejected, and user is asked if they want to place a new bet at the changed odds.
Response measure: 100% of bets affected by odds changes are safely rejected and re-confirmed, with user prompts issued within 1 second of detecting the change.

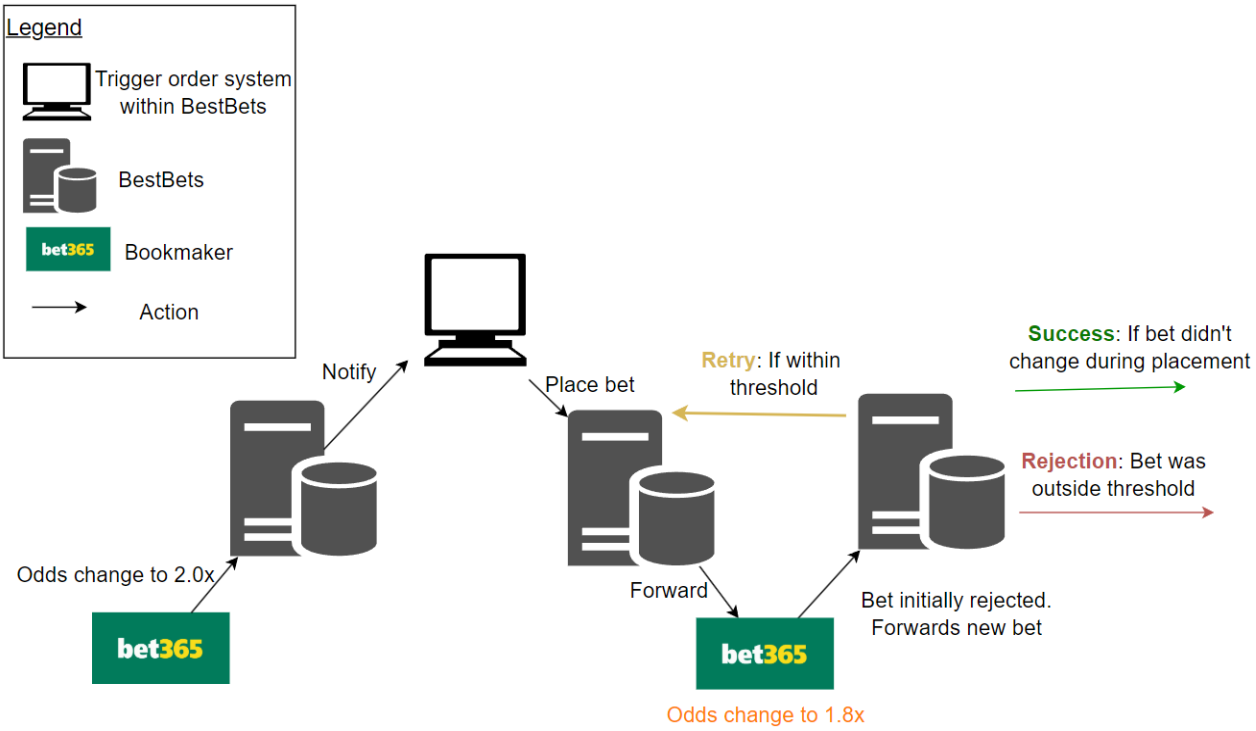


Figure 14: Informal diagram of safety scenario: Acceptance of changed odds. Logic is the same for user bets, except they must manually cancel.

Refined Scenario 2: Impossible bets can't be placed

Scenario: A user enters a desired bet that between confirmation and placement, gets removed from the bookmaker.
Relevant quality: Safety
Stimulus source: Bookmaker.
Stimulus: A user enters a bet and, immediately afterward, the bet is completely unavailable.
Environment: System is running normally.
Artifact: Module(s) handling placement of bets.

Response: The initial desired bet is rejected. User is warned that no bet has been made, and is presented the highest available bet from another bookmaker.
Response measure: 100% of bets affected by removal are safely rejected, with a warning issued within 1 second of detection.

Refined Scenario 3: External bookmaker errors does not cause faulty behavior

Scenario: A bookmaker is responding with error messages or not at all.
Relevant quality: Safety
Stimulus source: Bookmaker.
Stimulus: Error responses when trying to communicate with bookmaker.
Environment: System running normally.
Artifact: Bookmaker plugins.
Response: Our system continues to run without errors or faulty behavior.
Response measure: 0 error detected in our system in the case of errors from bookmakers.

3 Architectural synthesis

3.1 Module view

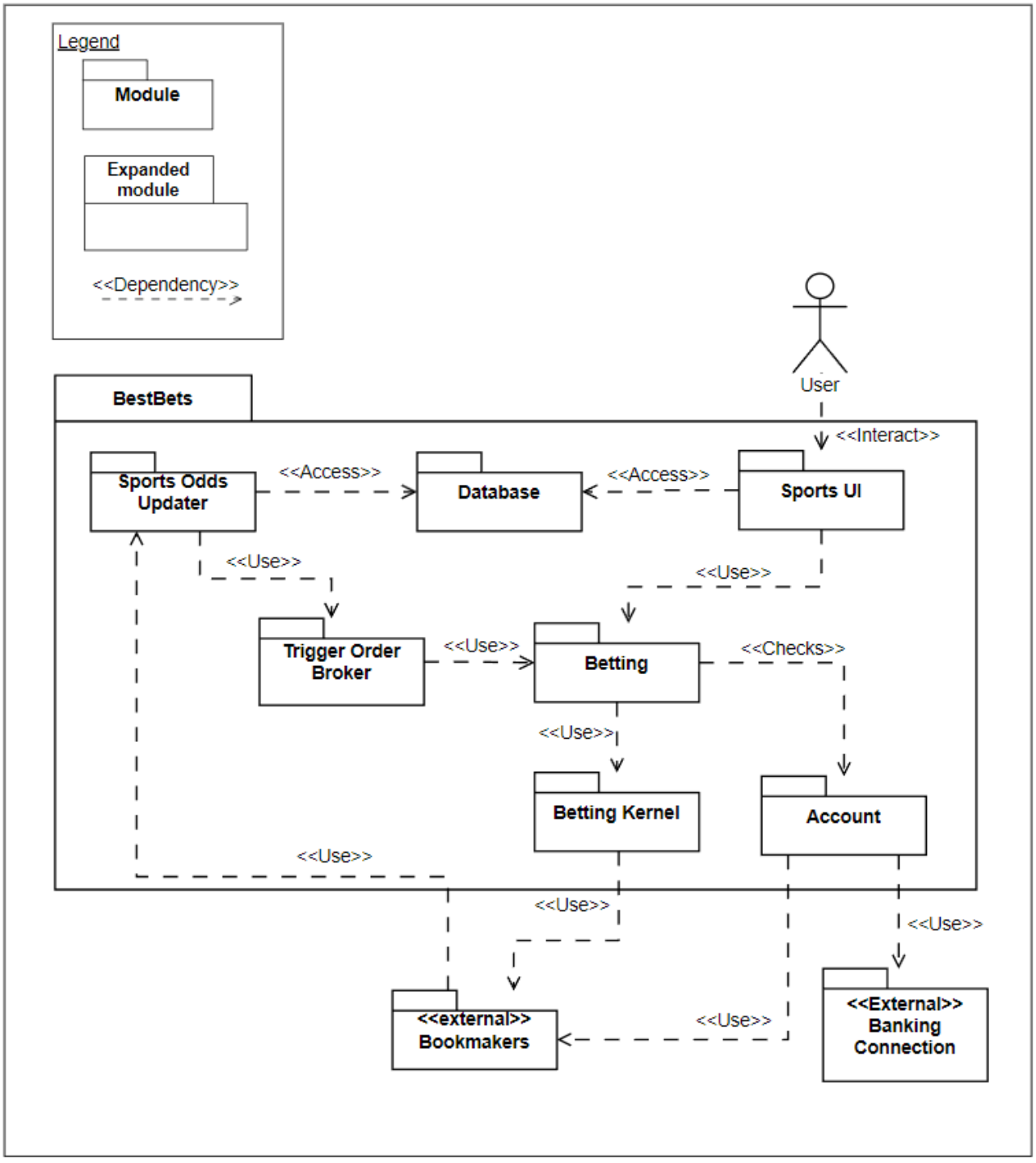


Figure 15: Module view of BestBets.

Account hold information about balance, it also has to confirm that the user is registered to the chosen bookmaker. (Fig 15)

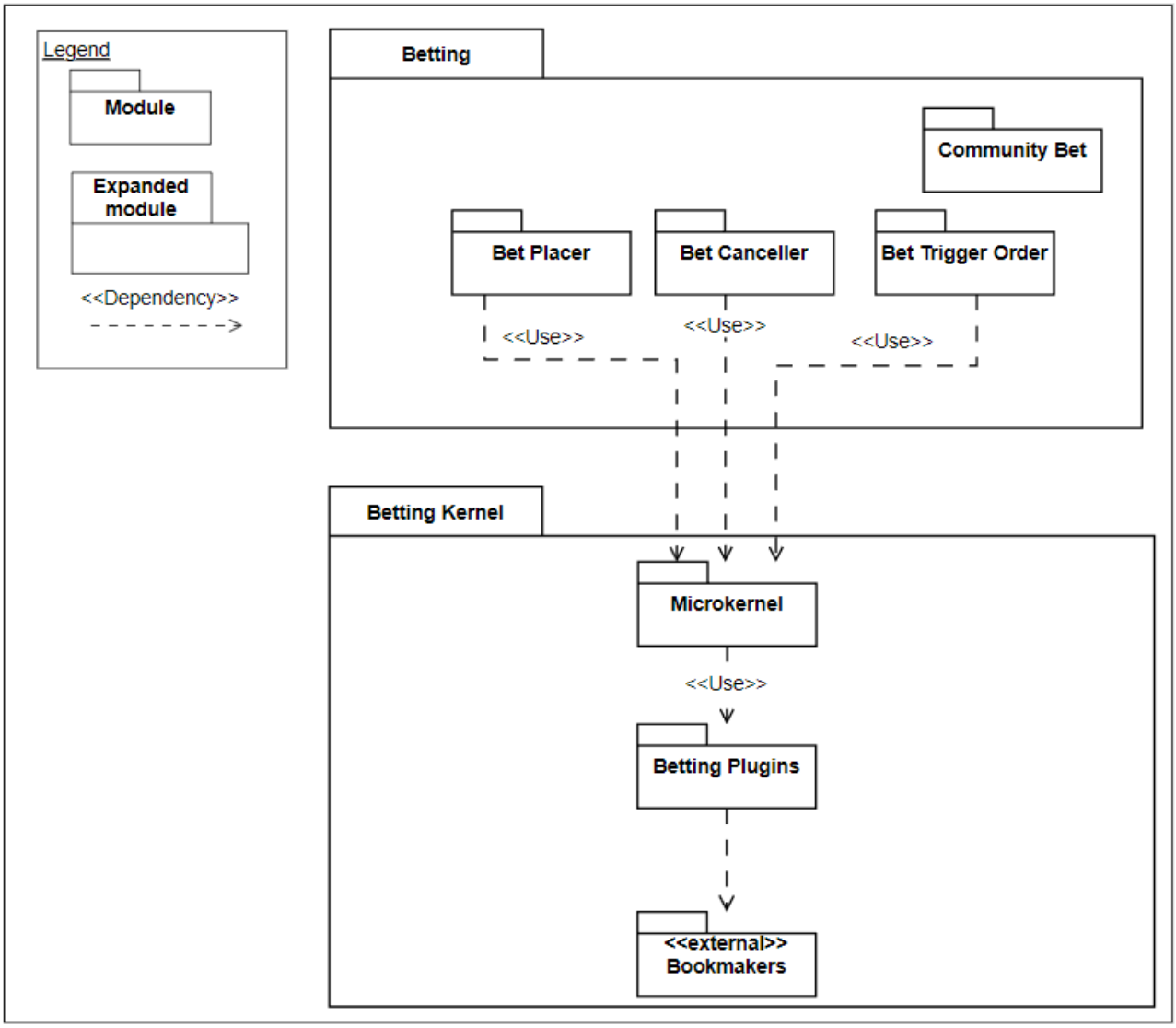


Figure 16: Module view of Betting and Betting Kernel modules.

Betting is what the user interacts with. They can place bets etc. Betting kernel handles all the behind the scenes logic of placing the bets at bookmakers. (Fig 16)

3.2 Component & connector view

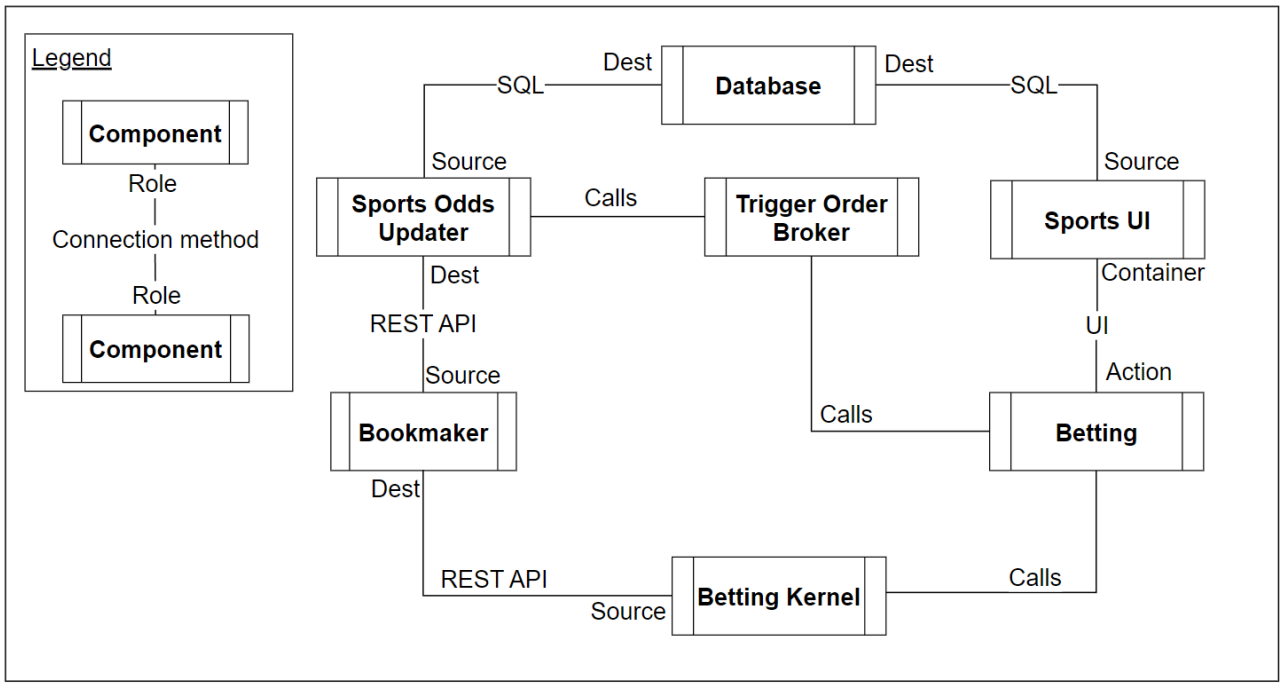


Figure 17: Component diagram of BestBets. There are still undecided connectors in the internal system. These are labeled as "Calls".

When bookmakers update BestBets through the Sports Odds Updater, the Trigger Order Broker informs all trigger order bets as a shortcut around the database. (Fig 17)

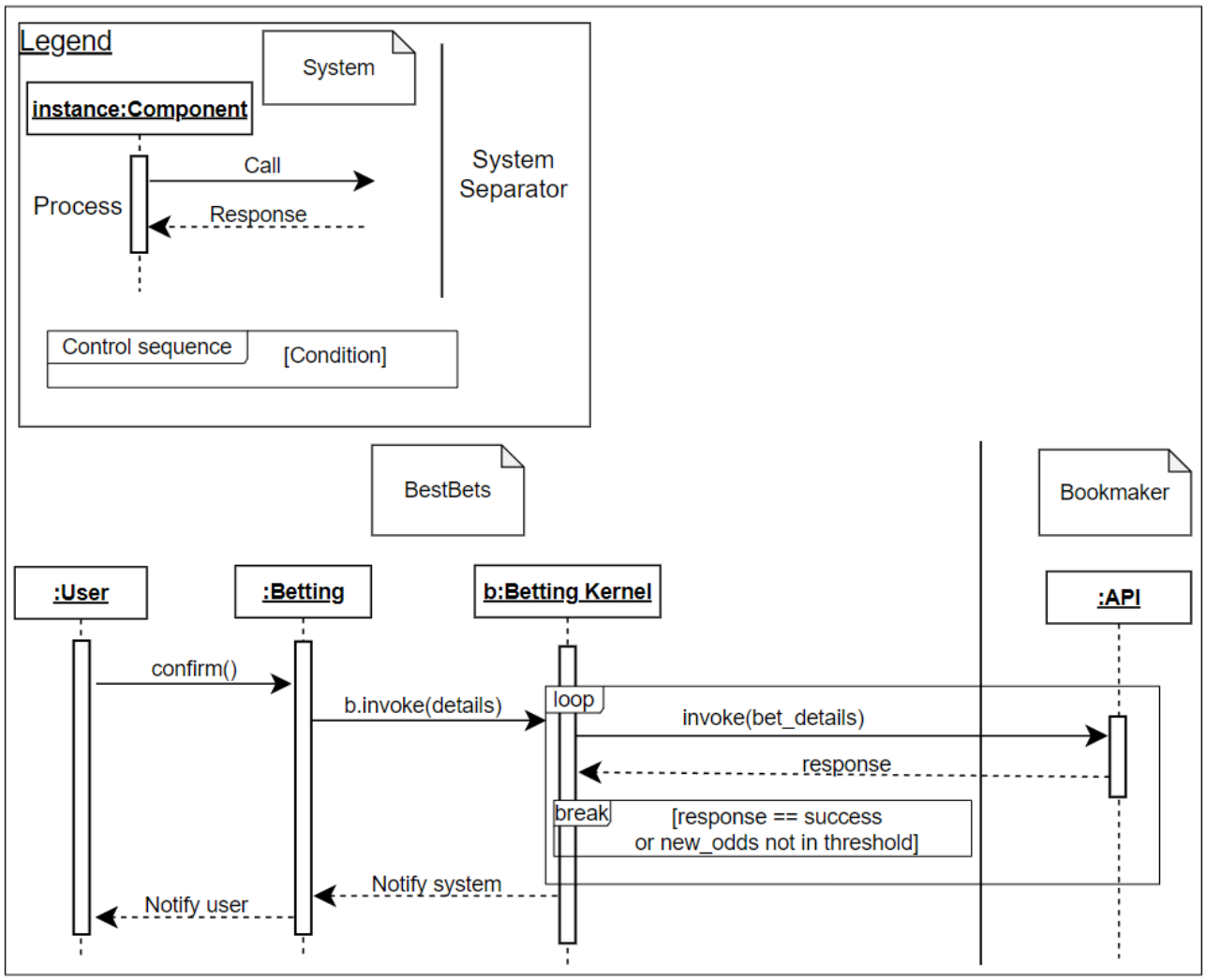


Figure 18: Sequence diagram of bet placement on BestBets.

When a user decides to place a bet on BestBets using the Betting component, it sends the request to the Betting Kernel which performs the transaction by repeatedly trying to place the bet on the bookmaker site until it succeeds or odds change beyond the users threshold. (Fig 18)

3.3 Deployment view

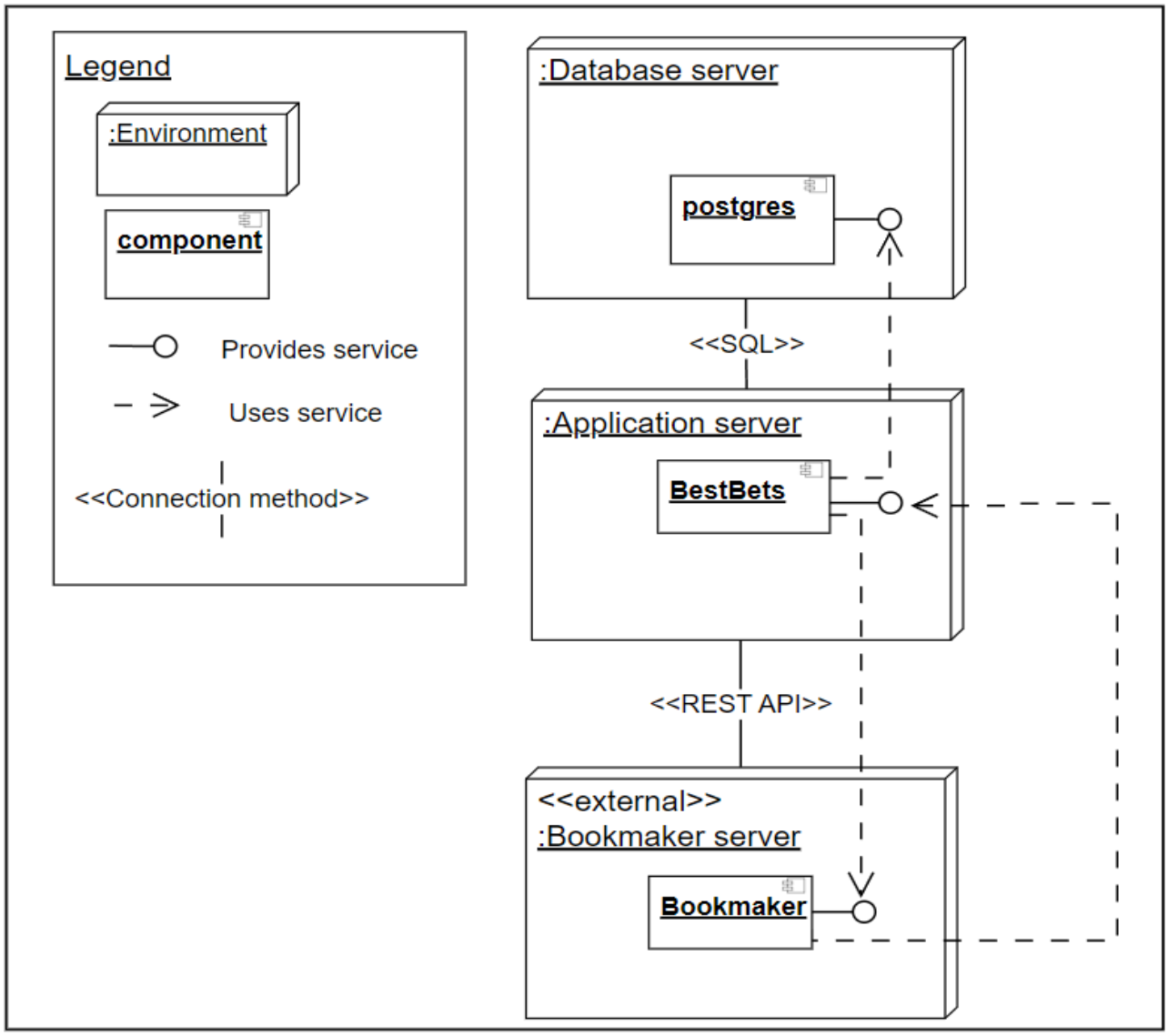


Figure 19: Deployment view of BestBets.

BestBets uses our database and talks to bookmakers. (Fig 19)

3.4 Design decisions regarding Tactics & Patterns

3.4.1 Overall structure

We will be zooming in on the following three components from Figure 20:

- Trigger Order Broker, marked with yellow
- Betting Kernel, marked with green
- Bookmaker plugins, marked with purple

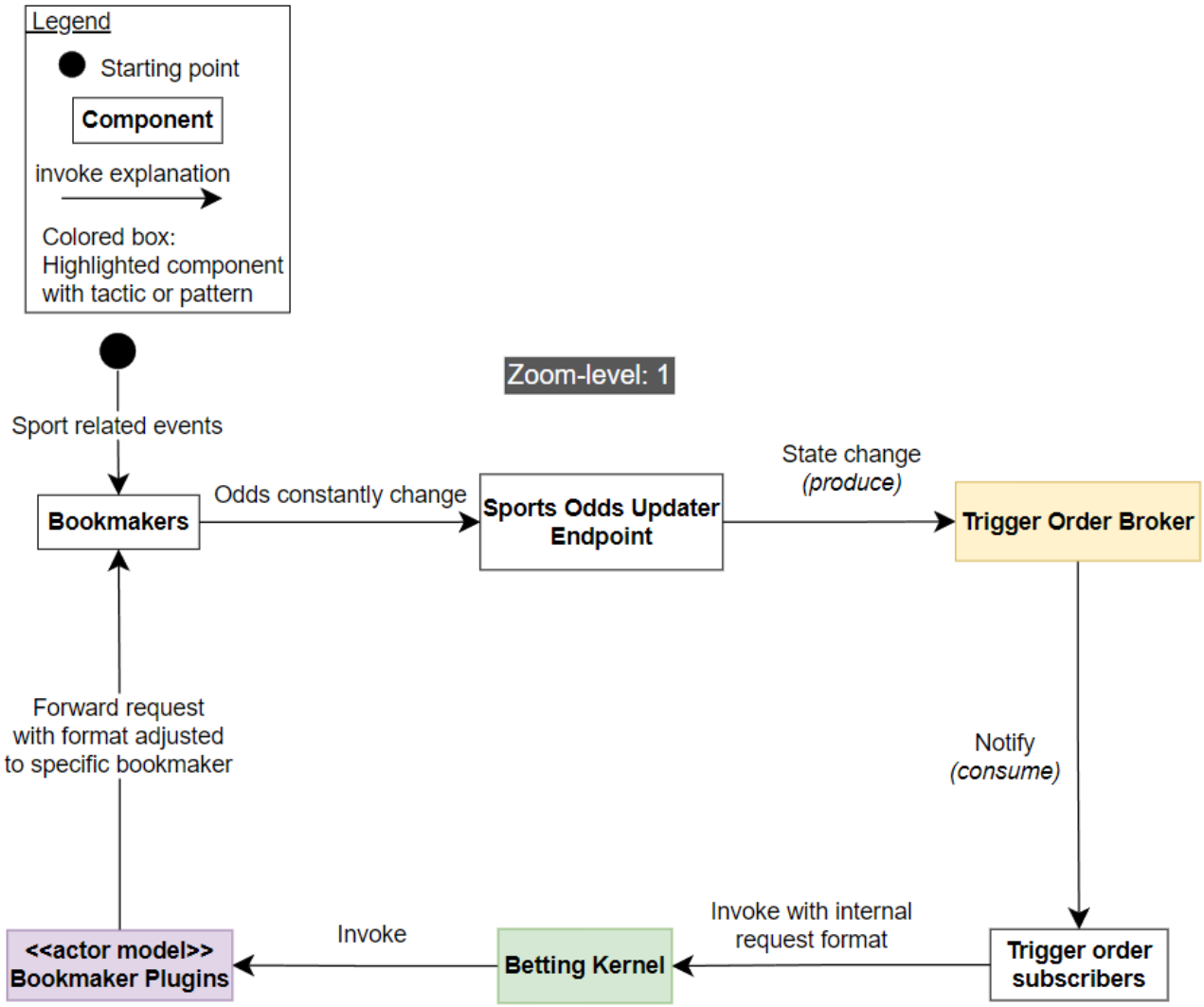


Figure 20: Overview of components involved in a trigger order bet.

3.4.2 Event broker zoomed in view

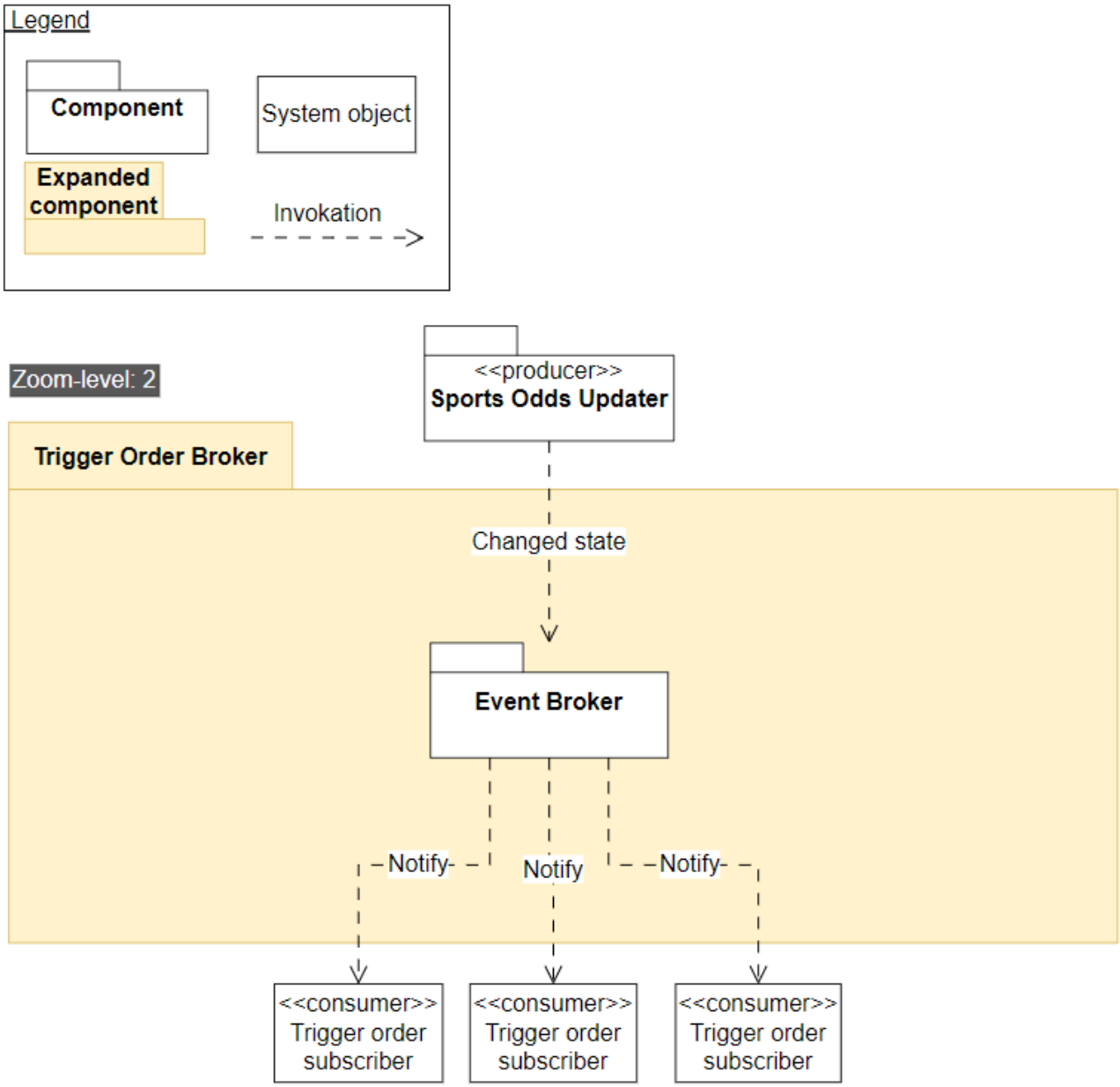


Figure 21: What happens inside the Trigger Order Broker.

The event broker pattern allows the Trigger Order Broker to quickly notify trigger orders (subscribers) when new odds are received from bookmakers through the Sports Odds Updater (producer). (Fig 21)

3.4.3 Microkernel zoomed in view

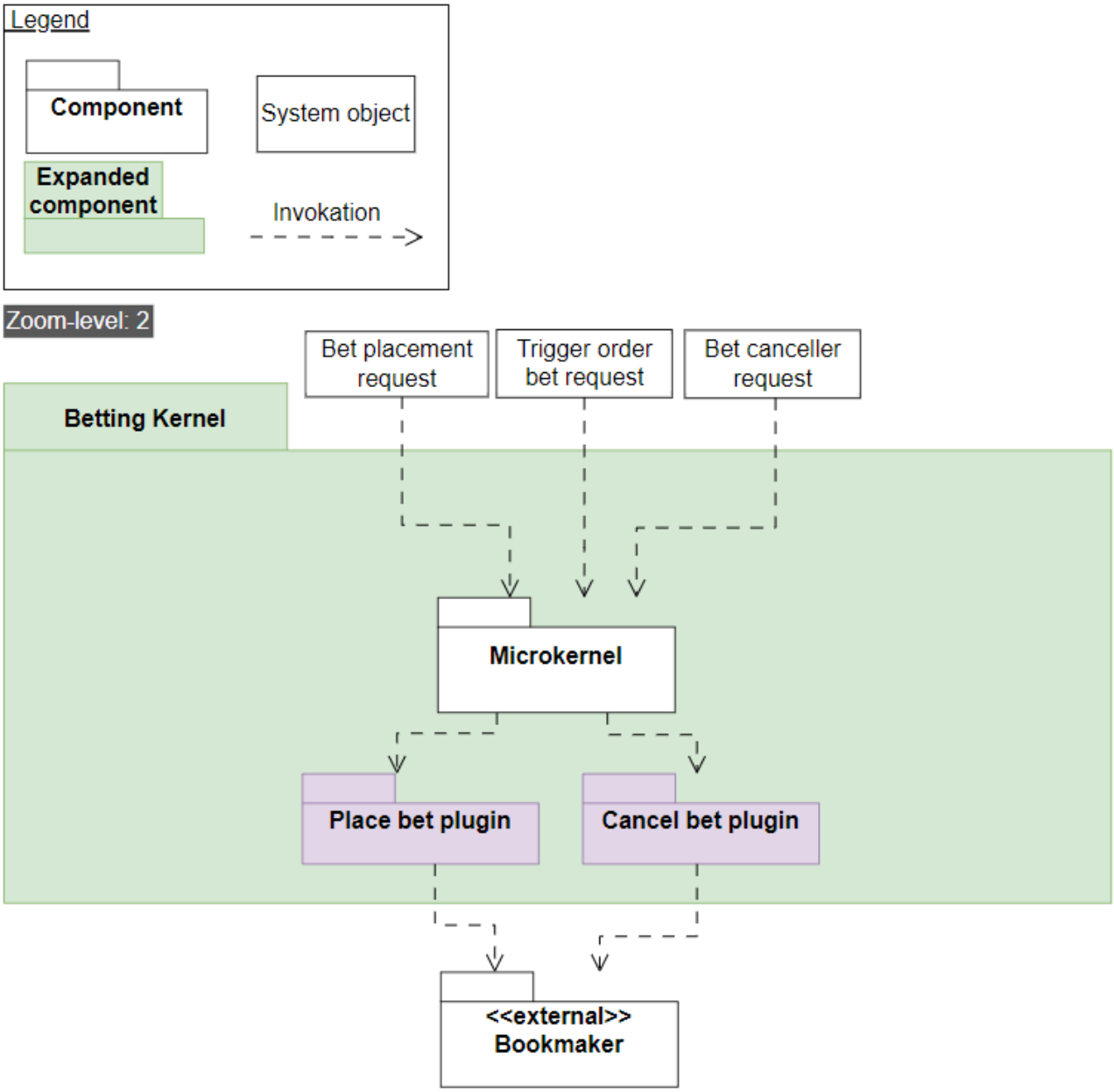


Figure 22: What happens inside the Betting Kernel.

The microkernel pattern is used inside the Betting Kernel to properly direct different kinds of requests to the proper handlers (plugins). (Fig 22)

3.4.4 Plugin actor model zoomed in view

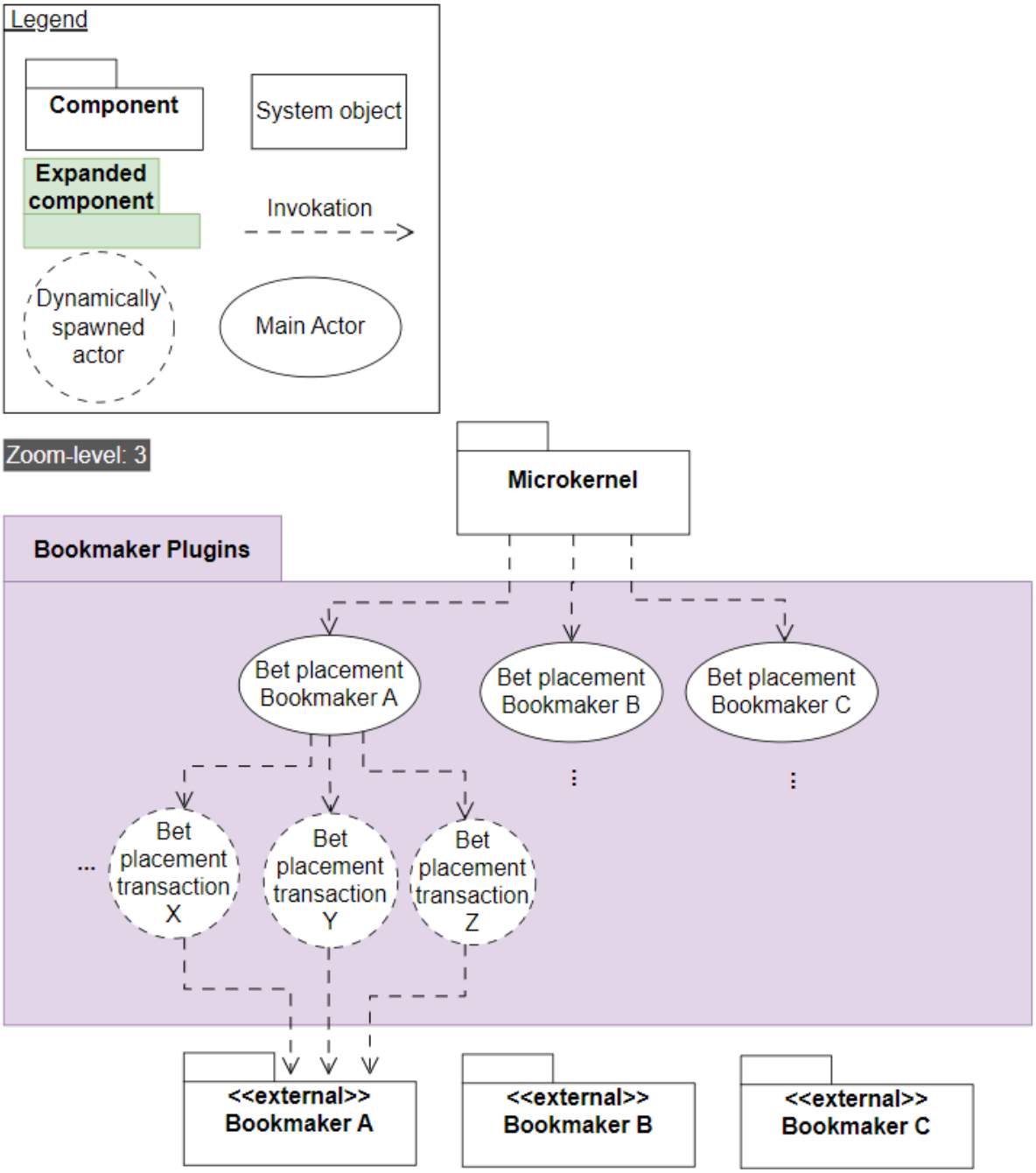


Figure 23: What happens inside the plugins that communicate with the bookmakers.

The Bookmaker Plugins use actor models to spin up temporary actors to perform single transactions for a specific bookmaker. (Fig 23)

3.4.5 Performance

3.4.5.1 Performance Patterns

Event broker (Fig 21)

Pros

- The event broker can allow for **low latency between odds being changed and the odds being displayed to our user**. Each channel in the broker will be a specific match ID. Trigger orders will then act as subscribers on specific channels with specific criteria being the odds and money placement.

Cons

- As the number of channels and subscribers increases, the broker **can struggle to efficiently manage all connections**, especially if channels are highly active. This can reduce performance.
- **Traffic might be heavily unbalanced**, as one channel might represent a local sports-match with a small audience and another represent the Champions League grand final match. This can reduce performance.
- It can be **difficult to ensure a correct ordering**. In case we end up deciding that bigger bets should be prioritized higher than small bets. This can reduce usability.
- The broker might be the **bottleneck and a single point of failure**. This can reduce availability.

Message Queuing using the Actor model (Fig 23)

Pros

- Spinning up an actor to handle each transaction ensures **greater resource utilization** as they can work in parallel.
- Decoupling from other bookmakers as **each actor is designed to only communicate with a specific bookmaker**. If the bookmaker changes the API, only a specific actor implementation is affected.
- **Each actor will function as a "bet transaction"**. If the bookmaker rejects the bet due to odds changing, the actor must retry if the bet is within the users threshold setting. This will enhance safety qualities.
- The actor model is suitable for **horizontal scaling** i.e. adding more "LeoVegas bookmaker actors", in case there is high traffic on bet placements for LeoVegas.

Cons

- Spinning up actors introduces some time overhead.
- Inter-actor communication can introduce additional latency, thus reducing performance.

3.4.5.2 Performance Tactics

Control Resource Demand, by **prioritizing certain bets**

Pros:

- Events like trigger orders working with **higher sums of real money should be prioritized** compared to someone betting with fun-money or placing a small bet below 100 kr. This can be achieved with the **event broker having a prioritization list**. This will improve usability for the customer segment more important to us (The whales) (Les Bernal, 2024).

Cons:

- Events like updating the UI might get higher latency.

Manage Resources, by **introducing concurrency**

Pros:

- **Process multiple streams of betting data in parallel**. This can be achieved with the actor model when handling bet placements. Each actor can act completely independent of other bookmaker actors, making their work “embarrassingly parallelizable”.

Cons:

- **Increased complexity** in ensuring we don’t accidentally spin up two actors for the same task and places 2 bets etc.

3.4.6 Availability

3.4.6.1 Availability Patterns

Actor model (Fig 23)

Pros

- The actor model can help preventing errors that causes the whole system to crash by **having the actors crash instead**. Having a monitor on the actors lets the system know so it can notify the user or try again.

Cons

- Lots of communication handling is needed, which may introduce computation and complexity overhead.

3.4.6.2 Availability Tactics

Detect Faults, by **Pinging**

Pros

- **Regularly check the availability of external bookmakers**. Unavailable bookmakers will be disabled on BestBets instead of causing a crash.

Cons:

- Increases network traffic and monitoring overhead.

Recover from Faults, with **Graceful degradation**

Pros

- When one bookmaker fails, **continue serving data from other bookmakers**. Achieved with actor model.

Cons

- Users may encounter reduced functionality in terms of bet placement options, which can lead to confusion or dissatisfaction.

3.4.7 Modifiability

3.4.7.1 Modifiability Patterns

Microkernel (Fig 22)

Pros

- Provides a handler that **supports placing/canceling bets on various external bookmaker API's**. This way if new bookmakers appear or change, we only have to handle the change in the microkernel logic while the rest of the system remains the same. Thus improving modifiability by **decoupling different bookmakers, and isolating the system affected by external changes**.

Cons

- Extra abstraction layers may lead to performance overhead.

3.4.7.2 Modifiability Tactics

Increase Cohesion, by **split module & Reduce Coupling**, by **encapsulating external calls**

Pros

- **Having different actors** (Fig 23) **for different bookmakers** increases cohesion and since they work independently they have low coupling.
- Letting the event broker (Fig 21) encapsulate by **handling odds changes on behalf of the whole system**.

Cons:

- Split module may decrease performance, because it adds more steps to the data-flow.
- Encapsulating in one place may introduce a bottleneck or single point of failure.

3.4.8 Safety

3.4.8.1 Safety Patterns

Actor model with message passing (Fig 23)

The actor handles the transaction process and **makes sure it either succeeds or quickly detects if odds have changed**, so the user can be notified. (See Figure 14 for more info)

3.4.8.2 Safety Tactics

Exception Detection, Exception Handling & Input validation

Pros:

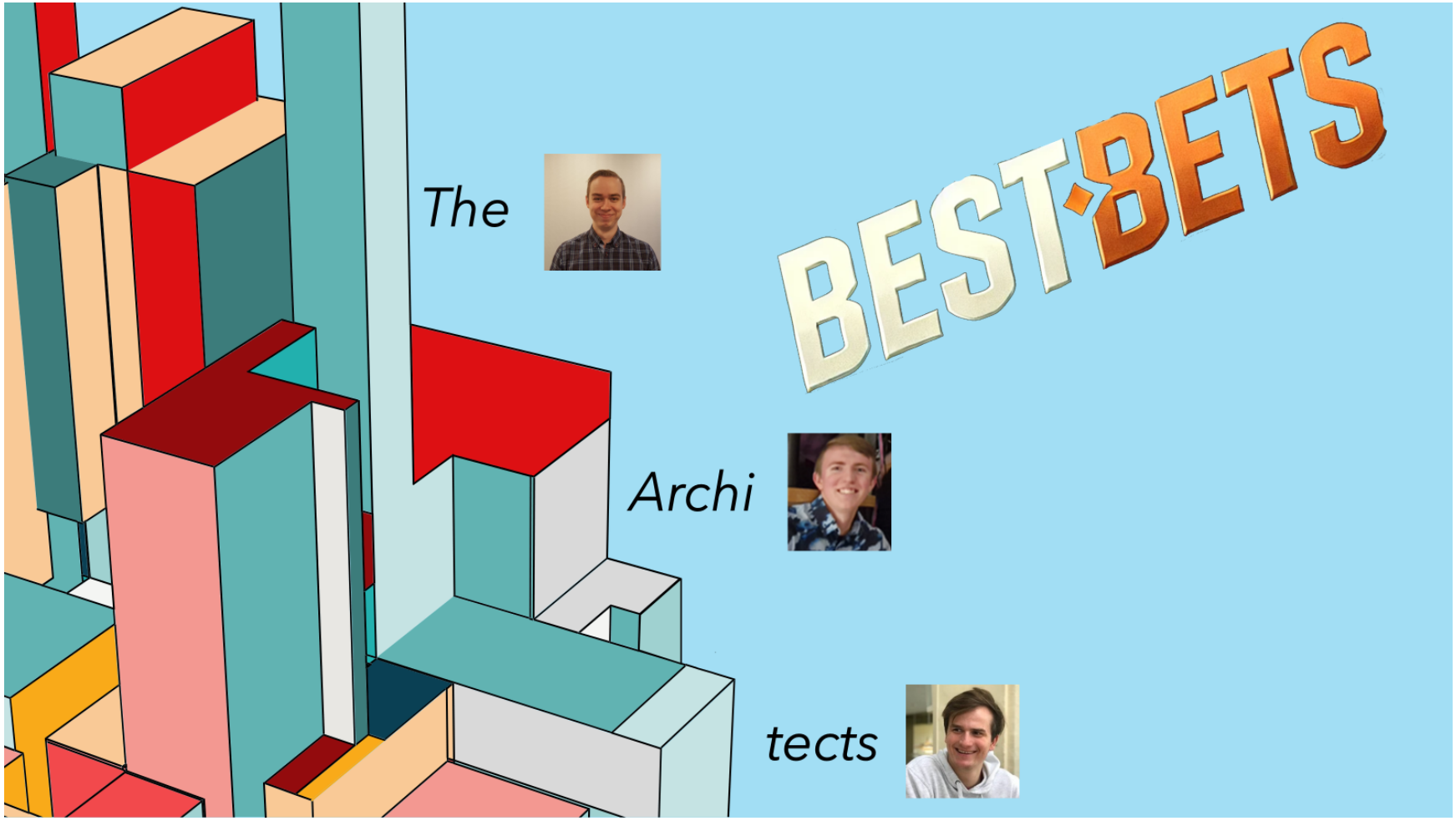
- Monitoring the actors to ensure crashes are handled, will avoid major faults.
- Immediately reject and rollback transactions if anomalies are detected.
- Making sure invalid bets cannot be placed by having the actors **closely communicate with the bookmaker API**.

Cons:

1. Additional processing may introduce minor delays.
2. Complex exception logic can decrease maintainability, by making it harder to make changes will maintaining a growing exception handling system.

4 Architectural Prototyping – Slides

We presented the prototype in class. These are the slides associated with the presentation.



AGENDA

PITCH

QAW HIGHLIGHTS

SYNTHESIS

PROTOTYPING

Q&A

PITCH

WHO HAS THE BEST ODDS?
- AND WHY ISN'T IT ALWAYS ME?

Random **sample of 7** audience members
prefered betting site,
for Tennis player A to win.

Odds	***Site***
1.98	NordicBet
2.07	betfair SPORTSBOOK
1.89	COMEON!
1.95	BETINIA
2.15	Betano
1.79	bet365
1.85	LeoVegas

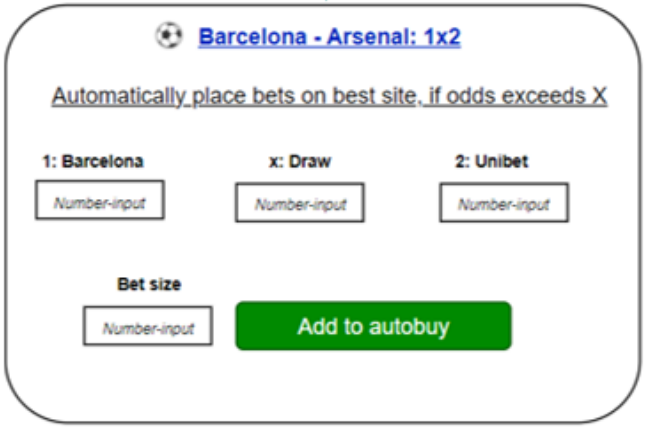
PITCH

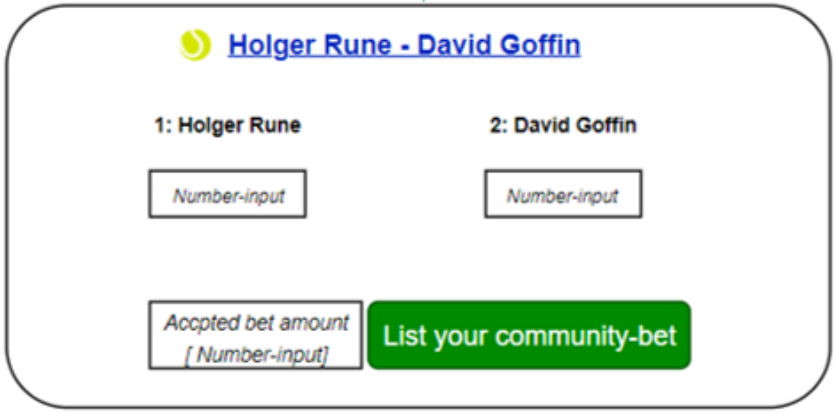
Core idea - Always shown best bets, from any bookmaker

Trigger-orders

Stock-like features

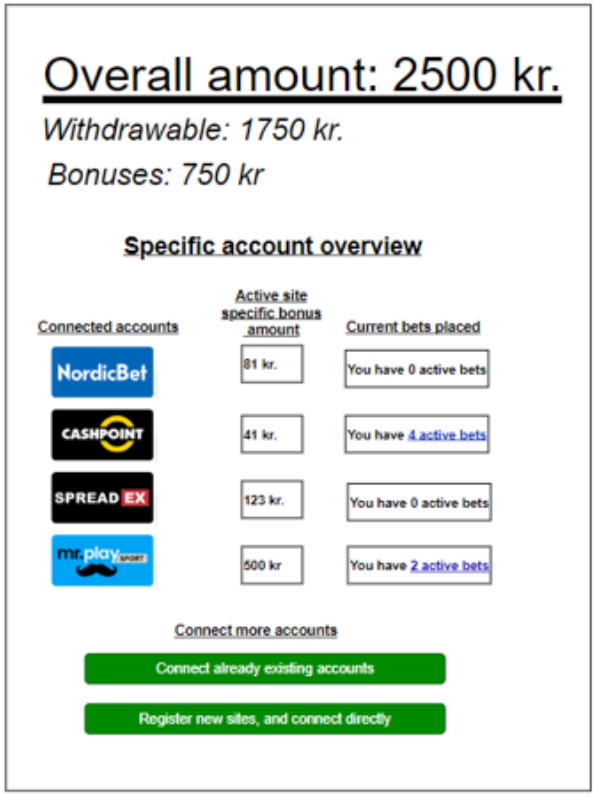
Be your own broker



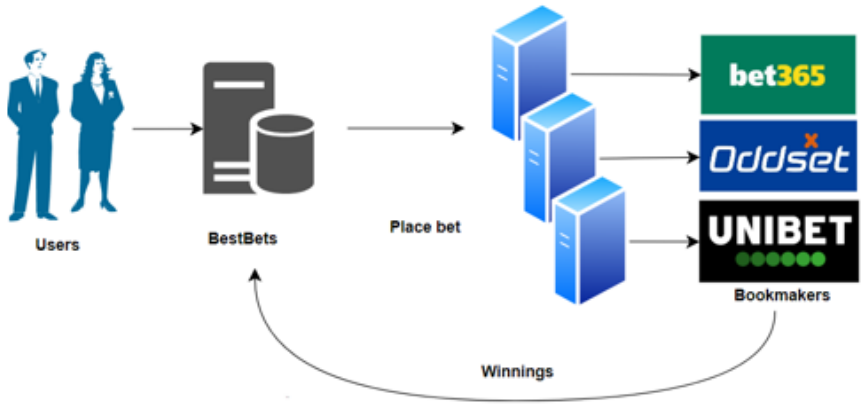


PITCH

Overview



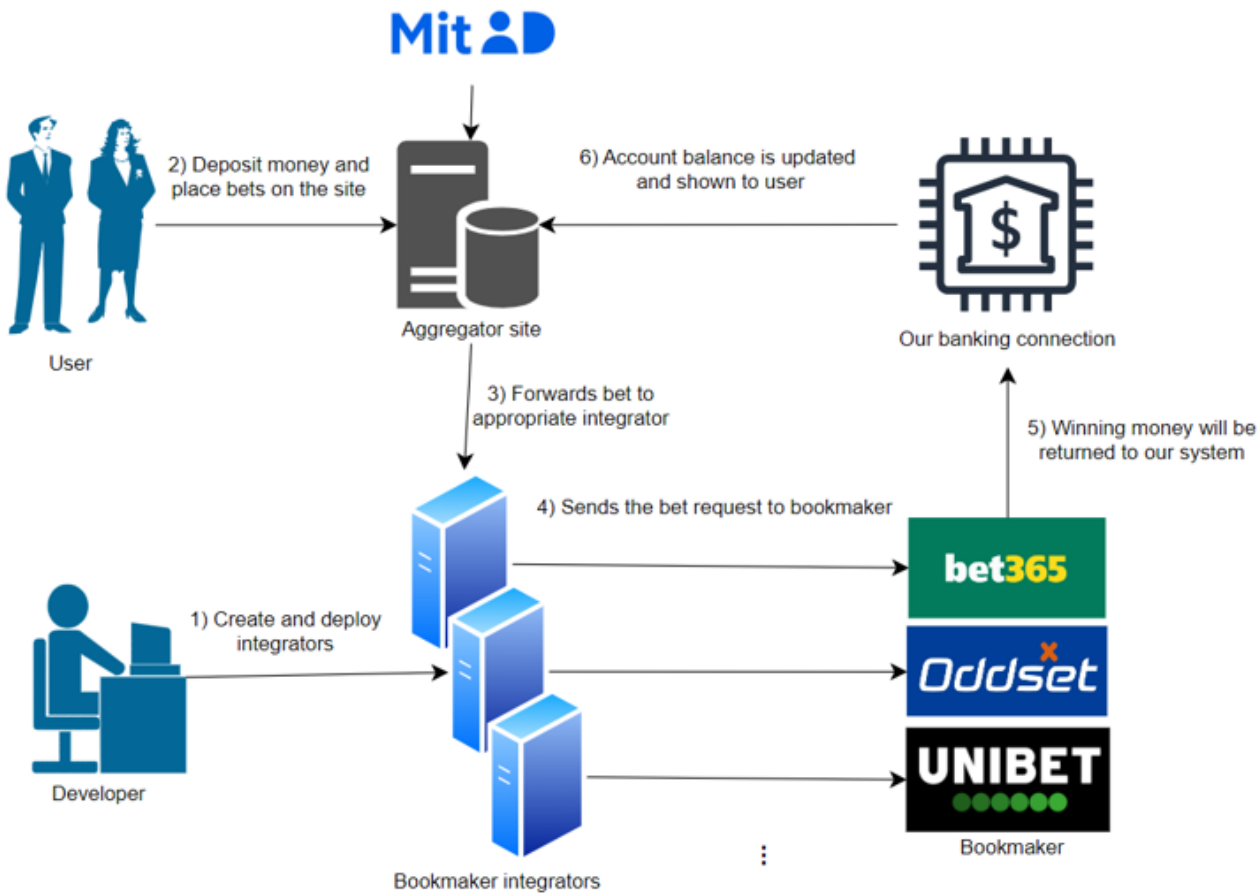
Centralized "banking"



Quickly register

PITCH SUMMARY

- Always get shown the best bets
- Only cash on a single site
- Consistent behaviour for every bookmaker



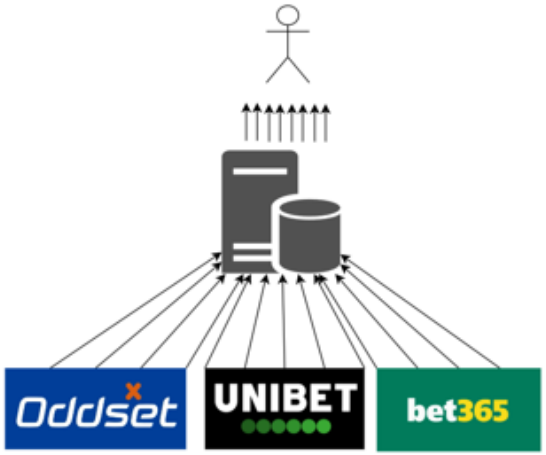
QAW HIGHLIGHTS

Quality	Performance	Modifiability	Safety
Scenarios	World cup • Many external bookmakers update their odds • Many BestBets logins • Many bets to forward	Bookmaker • Remove • Add • Changing API	Bets • Odds change before confirmation • Different thresholds • Impossible: removed from bookmaker
Votes	14	13	12

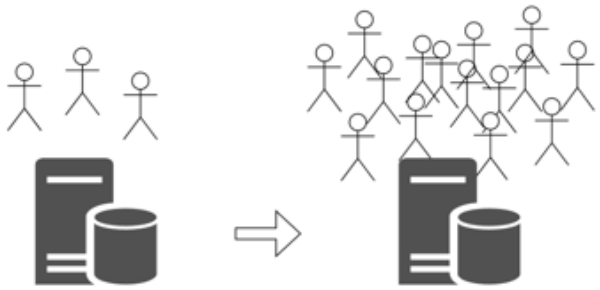
PERFORMANCE SCENARIOS & STYLES & TACTICS

Scenarios

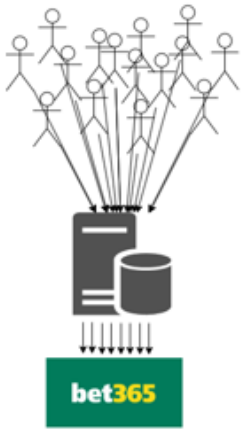
Real-Time Betting Odds Update



User login surge



Surge of bets



Pattern considered:

Event broker

Message queuing with the Actor model

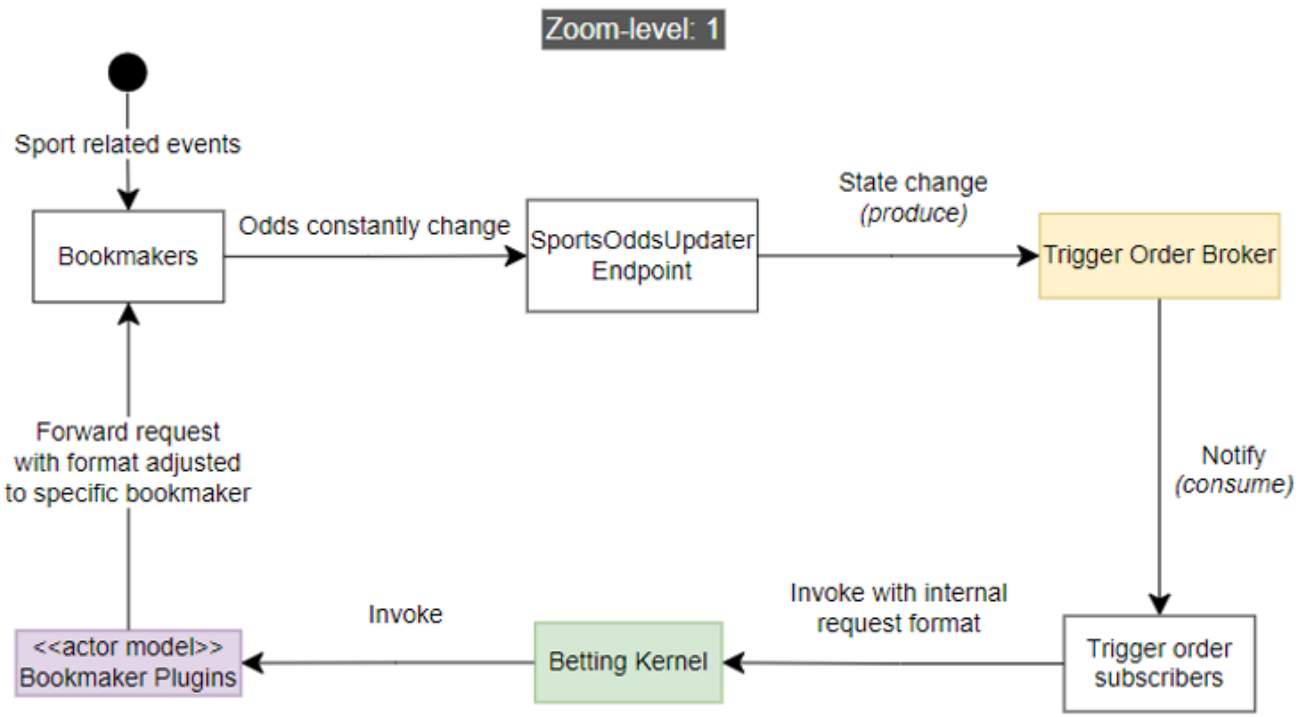
Tactics considered:

In general, Introduce concurrency:
Make the work embarrassingly parallelizable

Control resource demand:
Prioritize whales*

"86% of online gambling profits comes from 5% of the players." - Les Bernal ["Raising the stakes" (2024), CBC NEWS]

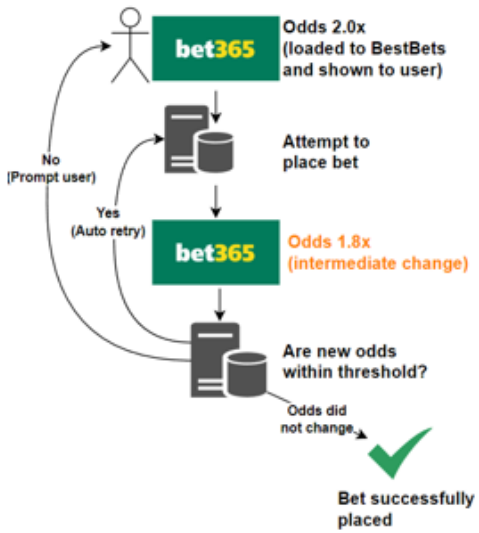
TRIGGER ORDER & BET PLACEMENT
zOOMIng IN ON THE PROTOTYPE



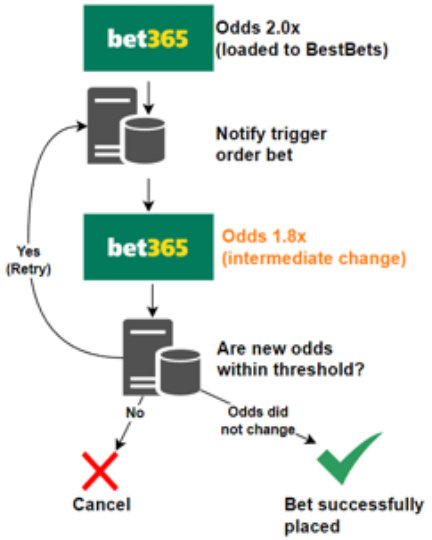
SAFETY SCENARIOS & STYLES & TACTICS

Scenarios

Non-trigger order



Trigger order



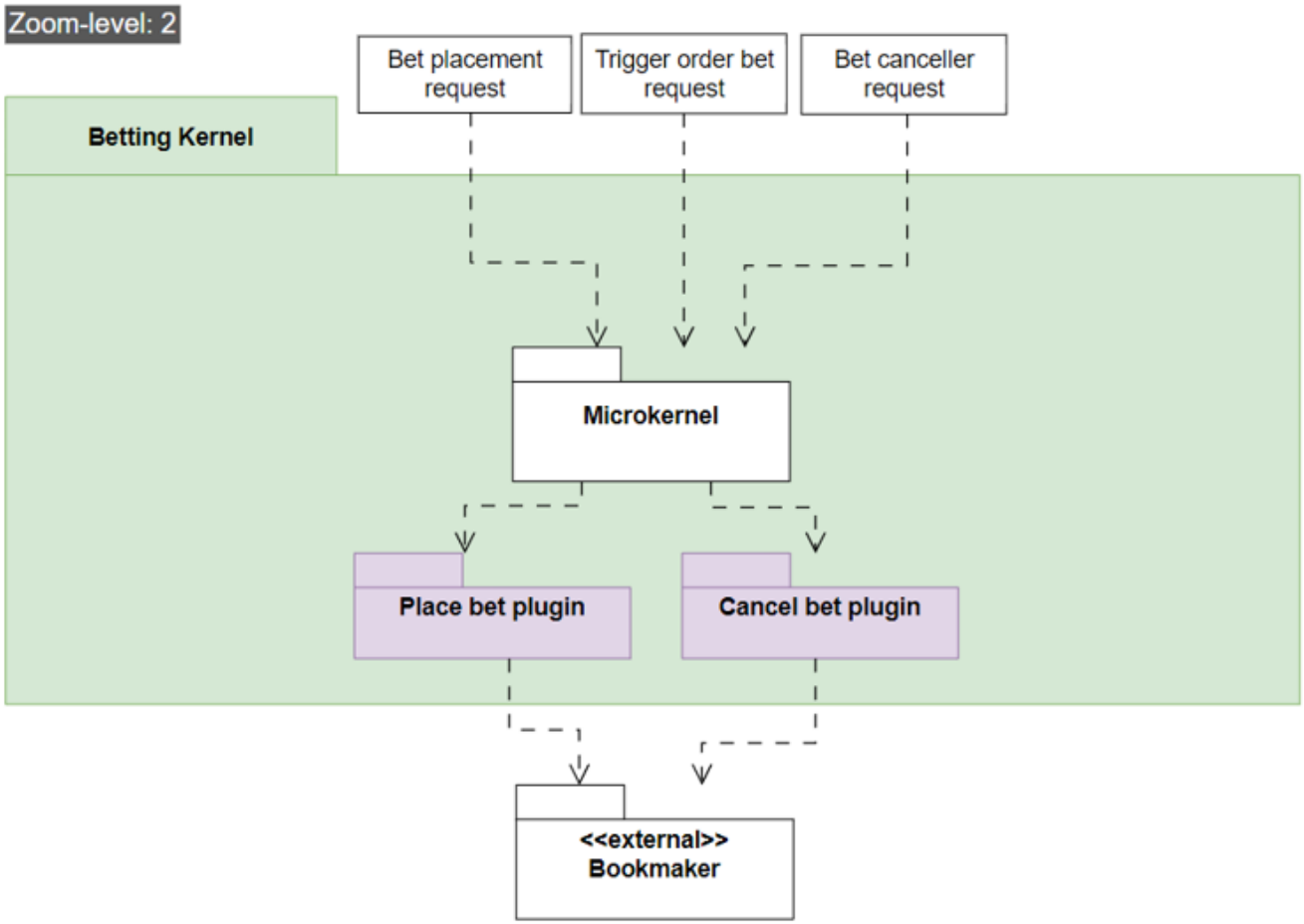
Pattern considered:

Actor model with message passing.
A single actor handles a complete **transaction** before being killed

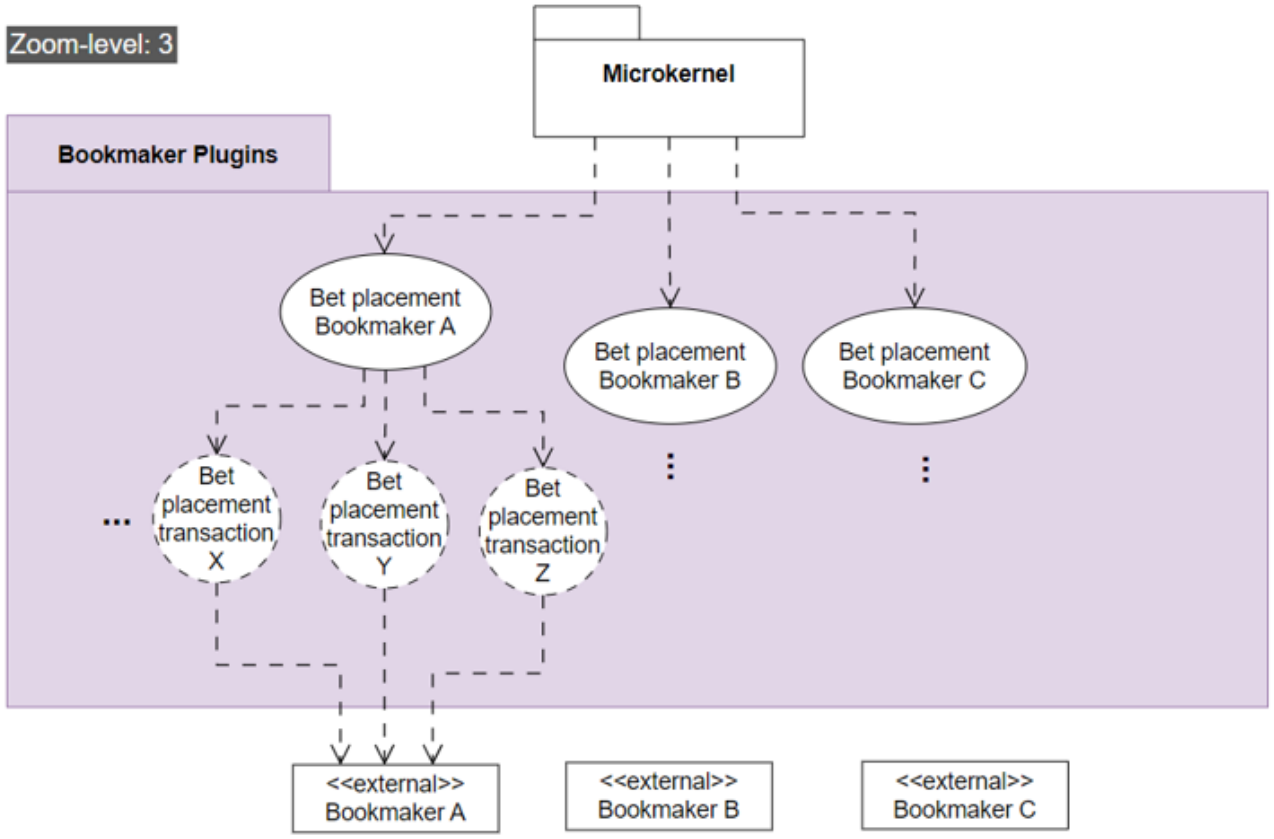
Tactics considered:

Transactions:
Consider **initial bet** placement as **start of transaction**,
And **final confirmation/rejection** as **end of transaction**.

MICROKERNEL ZOOMED IN VIEW



PLUGIN ACTOR MODEL ZOOMED IN VIEW



PROTOTYPING

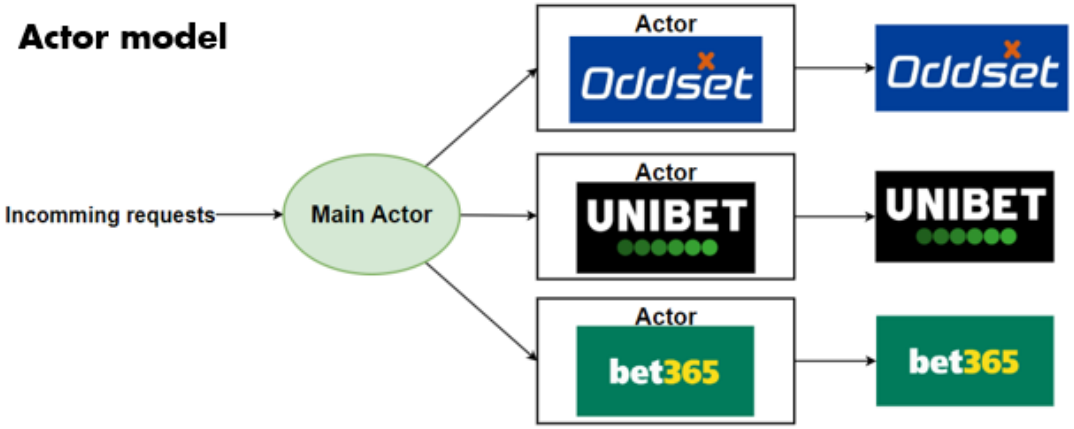
Concerns
- Is the system performant enough to handle world-cup-sized events?
- Can a bet transaction complete fast enough to get successfully safely placed?

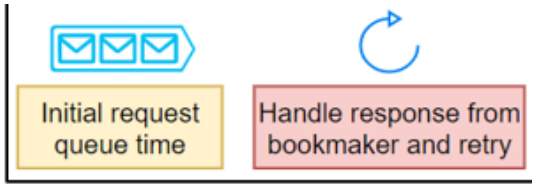
Hypothesis
- Performance
  The load balancer will perform better than the actor model.
- Safety
  The actor model will complete a single transaction with fewer retries than the load balancer.

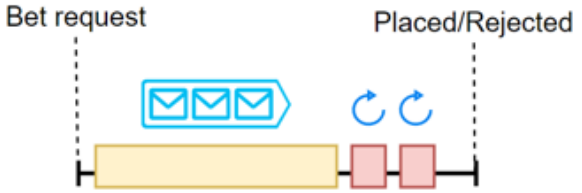
Actions to take during architectural prototyping
- Measure response time statistics.
- Measure average retries.

https://github.com/Grumlebob/Architecture/tree/main/Prototyping

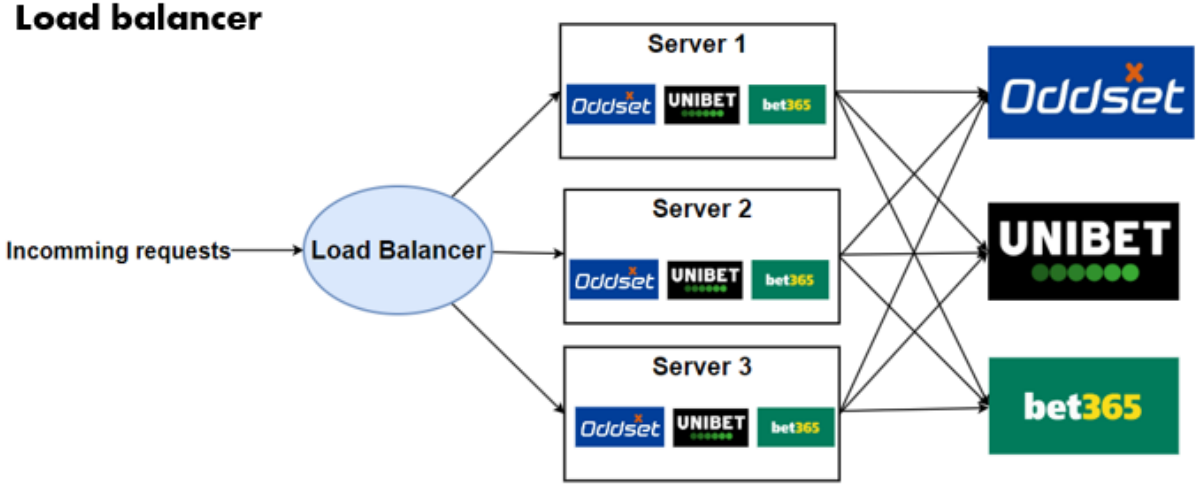
Actor model

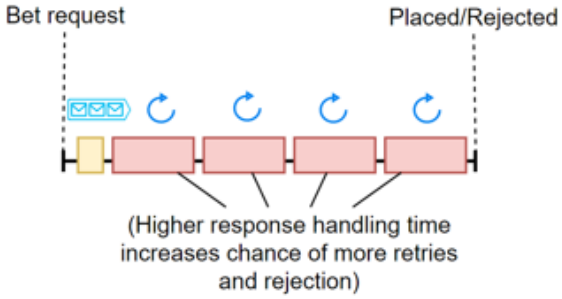






Load balancer





THREATS TO VALIDITY

- Prototype doesn't correctly simulate retries
- Every "bookmaker" had same logic. In real life bookmaker behaviour will vary greatly.
- Local vs real network

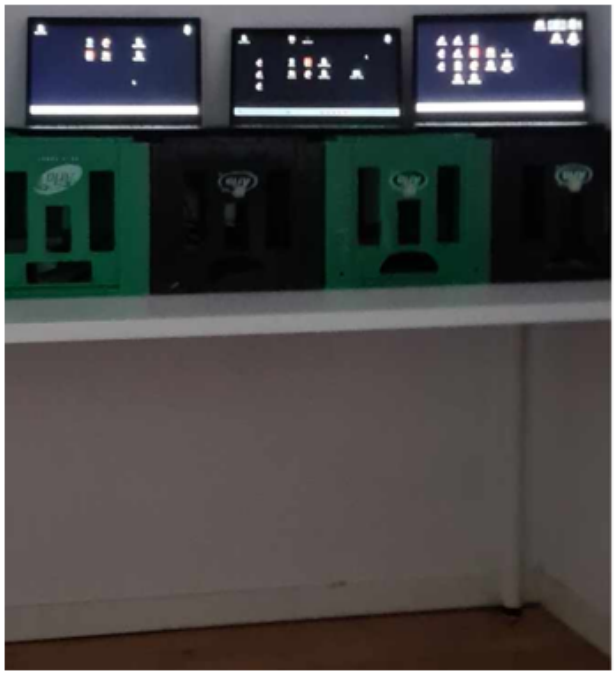
TEST SETUP

* *All are Lenovo Thinkpads T490*
* *All are on different networks*
  Courtesy of Grumlebet

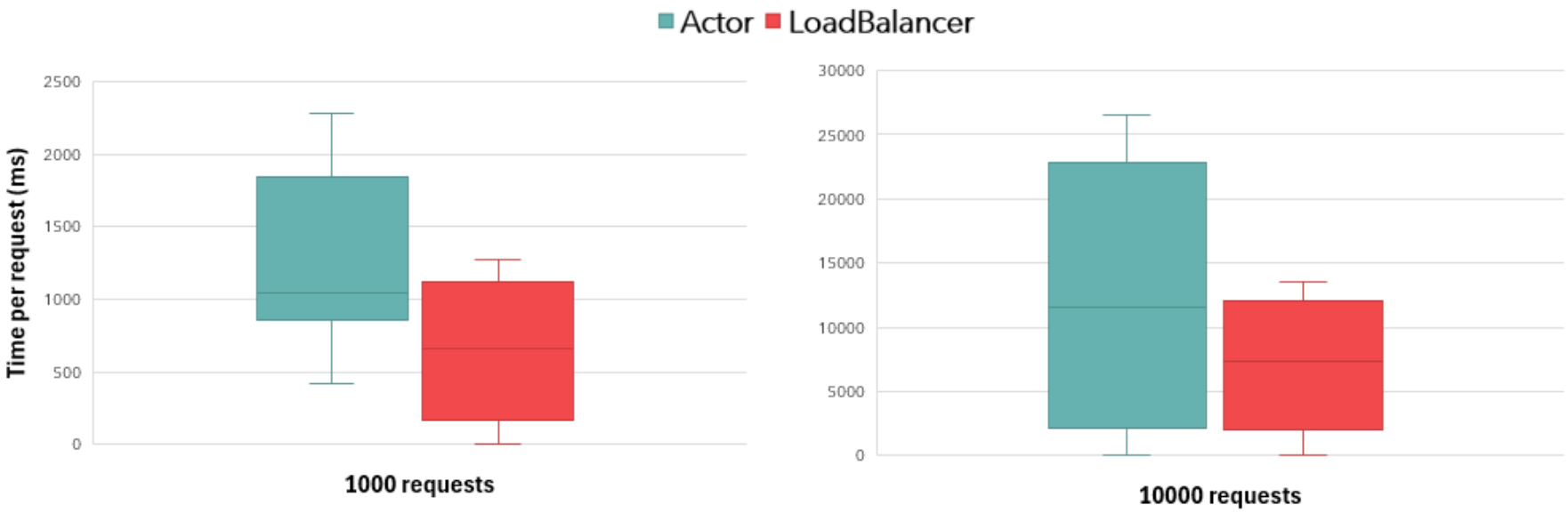
Multiple clients

Microkernel
Load balancer /
Actor forwarder

Multiple servers / actors



RESULTS



Actor model		Load balancer
− Lower throughput	Performance	Higher throughput +
+ Guaranteed single transaction prioritization	Safety	Complex prioritization −

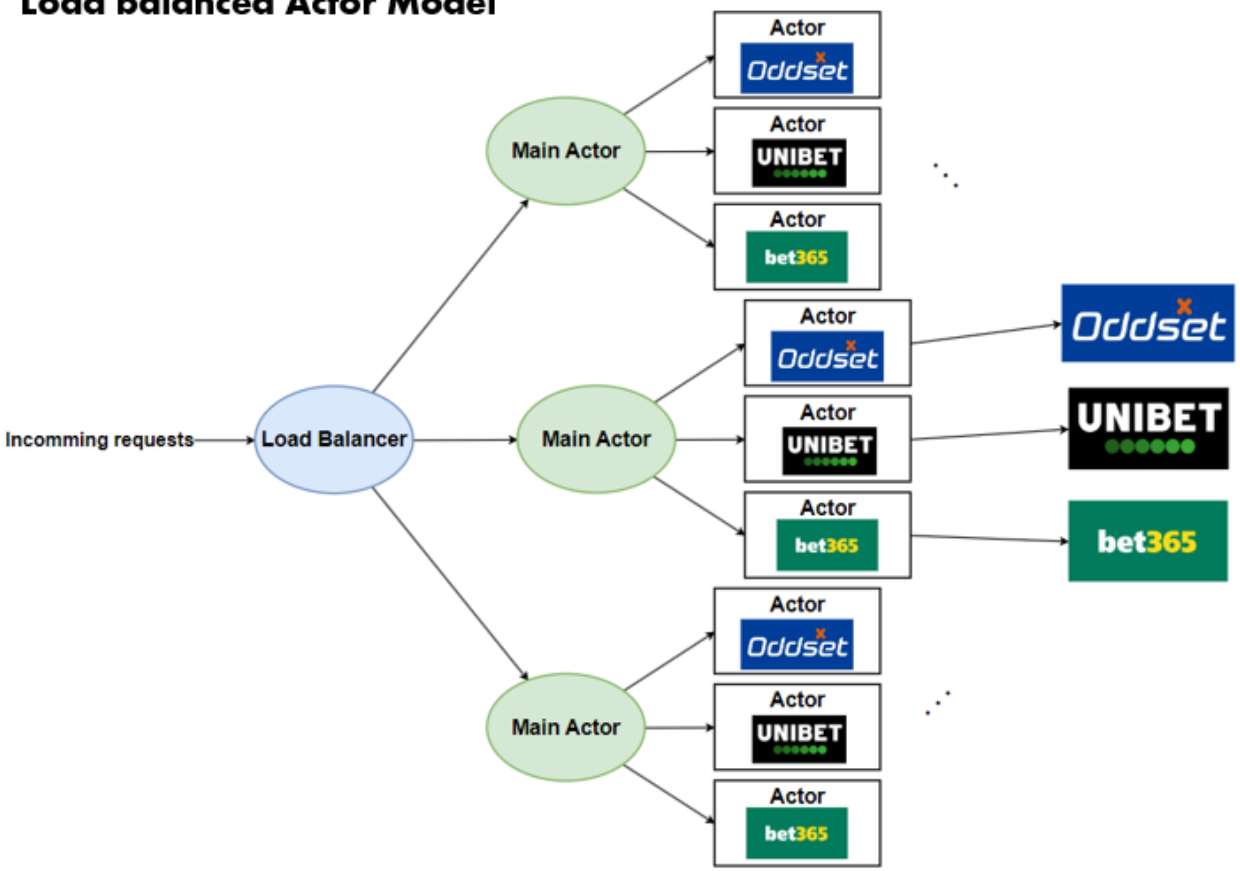
CONCLUSION

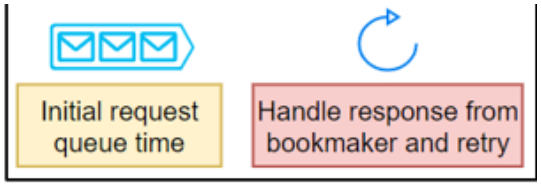
Perhaps a combination of the actor model and load balancer?

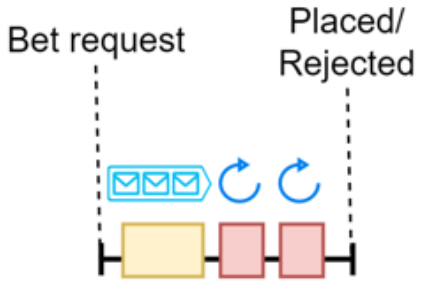
Pros:
- Load balancing to prevent bottlenecks
- Actors give natural separation and simple prioritization

Cons:
- Adds another step to the process which increases latency.
- Slight increase to modifiability complexity

Load balanced Actor Model







RELEVANT PIECES FROM THE LOG

- We want to zoom in on the event broker as well and make a prototype
- We want to look more in-depth into the load balanced actor model

THANK YOU

Questions?

5 Architectural Evaluation (aSQA Method)

For the final evaluation of BestBets' architecture, we chose the **aSQA** method. **It is well-suited to our agile iterative process**, with constant new lectures, feedback and learning, because it quantifies quality attribute levels and identifies focus areas with relatively little overhead.

We defined target levels and current levels (based on our design and prototyping results) on a scale where 1 = Unacceptable, 3 = Acceptable and 5 = Excellent (Christensen et al., 2010).

5.1.1 Components for the first aSQA iteration

Note: After the evaluation we ended up modifying some components to increase their scores and went back and updated the report document. The old components are introduced below because they are relevant for the first iteration of the evaluation. For the Betting Kernel we had two different proposals that we tested against each other. They serve the same functional purpose. This was explained in the presentation as seen in section 4.

We use the three most important identified quality attributes, found in our architectural analysis. For the **components we have selected** the ones we have gone most in-depth with during this project, **being the key features of the entire system**, namely placing bets and receiving odds updates.

- **Global Odds Broker**
  It is notified by the odds updater endpoint that external bookmakers uses, and then distributes to relevant trigger order consumers, as well as relevant UI pages that shows odds that are now updated.
- **Actor model (Microkernel)**
  Where a single actor server only speaks with a single bookmaker.
- **Load balancer**
  Where a single server speaks with every bookmaker.

5.1.1.1 Performance

Component	Target	Current	Health	Importance	Focus
Global Odds Broker	3	1	3	2	2
Actor Model	5	2	2	5	4
Load Balancer	5	5	5	5	1

Table 1: *Performance results of the first aSQA iteration*

Discrepancies can occur between the bookmaker odds and the odds visible in our UI. The actor model / load balancer uses the odds from the bookmakers when doing an actual bet placement, and as such have a higher importance than the Global Odds Broker (Table 1). This is ordinary in the stock-broker market, where most free UI's have up to 15 minutes delay of the actual price (Børsen Investor, 2025).

5.1.1.2 Modifiability

Component	Target	Current	Health	Importance	Focus
Global Odds Broker	2	1	4	3	2
Actor Model	4	5	5	4	1
Load Balancer	4	4	5	4	1

Table 2: *Modifiability results of the first aSQA iteration*

Both the actor model and the load balancer use a microkernel pattern, which separates concerns and simplifies adding and removing bookmakers, resulting in overall good scores. The actor model has a more natural way of separating concerns, which is not necessarily true for the load balancer, hence the slightly lower current score.

5.1.1.3 Safety

Component	Target	Current	Health	Importance	Focus
Global Odds Broker	2	1	4	2	1
Actor Model	4	4	5	5	1
Load Balancer	4	1	2	5	4

Table 3: *Safety results of the first aSQA iteration*

The Global Odds Broker is not in charge of doing the actual money placements and following transactions resulting in a lower target/importance score. In contrast the actor model/load balancer, handles actual placements of money and therefore have high target/importance scores. The load balancer handles retries by having requests reenter the queue, which harms the current safety score.

5.1.2 Summarized evaluation of first iteration

In general the Global Odds Broker have low current scores because we haven't prioritized it yet. The focus has so far been on the part of the system that actor model and load balancer can handle. Their importance scores are also higher because this is the part of the system where the key feature of our concept is implemented: Being able to bet on all bookmakers from one centralized place.

5.1.3 Changed components for the second aSQA iteration

Based on the previous iteration we identified that the components could use slight changes and more in-depth specifications. Below are the changes made to the components in an attempt to improve their scores for each of the three quality attributes:

Global Odds Broker → Trigger Order Broker
No longer notifies the UI. How the UI gets the latest odds is currently undecided and placed as the highest priority in the backlog.

Load Balancer / Actor Model → Load Balanced Actor Model
Bet requests are now passed through a load balancer and distributed to servers with actor models as shown in section 4.

5.1.3.1 Performance

Component	Target	Current	Health	Importance	Focus
Trigger Order Broker	3	2	4	2	1
Load Balanced Actor Model	5	5	5	5	1

Table 4: *Performance results of the second aSQA iteration*

The Global Odds Broker is now less likely to become a bottleneck with its reduced responsibilities.

5.1.3.2 Modifiability

Component	Target	Current	Health	Importance	Focus
Trigger Order Broker	2	1	4	3	2
Load Balanced Actor Model	4	3	4	4	2

Table 5: *Modifiability results of the second aSQA iteration*

The load balanced actor model has more complexity and now requires each server to have a copy of the same actor model, hence a lower current score.

5.1.3.3 Safety

Component	Target	Current	Health	Importance	Focus
Trigger Order Broker	2	1	4	2	1
Load Balanced Actor Model	4	4	5	5	1

Table 6: *Safety results of the second aSQA iteration*

Given the load balanced actor model is a combined solution, it preserves the safety from actor model. This directly improves the relatively low safety of just the load balancer from the first aSQA iteration.

6 Conclusion

In this project we have worked on an architectural design of a system allowing users to conveniently place bets on sports matches from a single centralized source called BestBets.

Our current architectural design still needs a lot of work. Up until now our focus when prototyping has been on the betting kernel, the system that handles placing bets. We started with a proposal of using an actor model, but after testing it against a load balancer we realized it needed to be optimized for better performance.

We still need to decide how the database should be designed, specifically if it should be a single instance or if we want replicas with read-only data. The same goes for other components like how the UI should fetch odds data, and how odds for a general page with sport matches are being updated.

Despite not having a complete design ready to be fully implemented, the selected key-components were complex enough to provide us a great opportunity to apply the core theory of the course. Thus the end result was satisfactory for us and we learned a lot from this course.

7 References

- Børsen Investor. (2025). “Dit Dybdegående Aktieunivers”. https://borsen.dk/investor/kurser/danske-aktier/. Accessed 12 May 2025.
- Christensen, H. B., Hansen, K. M., & Lindstrøm, B. (2010). “Lightweight and continuous architectural software quality assurance using the aSQA technique”. I M. A. Babar, & I. Gorton (red.), Software Architecture: 4th European Conference, ECSA 2010, Copenhagen, Denmark, August 23-26, 2010. Proceedings (s. 118-132). Springer. https://doi.org/10.1007/978-3-642-15114-9_11
- Spillemyndigheden. (2025). “Velkommen Til Spillemyndigheden”. https://www.spillemyndigheden.dk/. Accessed May 12, 2025.
- Spillemyndigheden. (2025). “Licence holders”. https://www.spillemyndigheden.dk/tilladelsesindehavere?field_license_type_target_id=All. Accessed 16 May 2025.
- Les Bernal. (2024).“Raising the Stakes”. CBC News. https://www.youtube.com/watch?v=Pxvfy4qQRog&t=1070s. Accessed 16 May 2025.

8 Architecture backlog

8.1 Our guide to our logs

1. Always remember to set dates.

2. Some idea might fit multiple logs/headers. It is not super important which is chosen, what is important is that it is written down.

3. Avoid merge conflicts of this Libre Office file by:

- If alone, write it down in your own local file first, then meet with group to get it merged.
- If with group, ensure only 1 is changing the log, then commit updates and notify group, so no concurrency issues arise.

4. If the idea is heavily conflicted by the group. Still write it down, but note down the disagreement.

5. Colors:

- Green: Finished / implemented / done
- Black: Being worked on / being discussed
- Red: Turned out to be irrelevant / discarded

8.2 Latest overview of top chosen items from backlog

Updated: 16.05.2025

1. We need to make **every decision regarding database**. Example: Do we have one database that external bookmakers update, and then another read-only replica database that the sports UI uses?
2. We also need to make **every decision regarding updating the user and account information**, sending bet confirmations, etc. In short, all the work that happens AFTER a bet has been successfully made and received back from the external bookmakers.
3. We need to make **decision regarding user settings**, such as setting odds-variance-thresholds.
4. We need everything regarding **specific site-relevant information on specific bookmakers**, such as how **check** of **a bonus on another bookmaker**, happens already in our UI **before placing the bet**.
5. We want to look more **in-depth into the load balanced actor model design**.
6. We want to **zoom in on the event broker** as well and make a prototype.

8.3 Ideas related to business

8.3.1 Fictitious project

10.02.2025

We have made some fictitious assumptions, to make this project interesting to work with.

- All danish bookmakers have agreed to be part of BestBets.
- All bookmakers already have some API for placing/canceling bets.
- There are no legal troubles, such as not complying with various laws from Spillemyndighederne
- Transfers can be made between various bookmakers, and our own bank, as to avoid the restriction of “transfers must be through a MitId provider og NemKonto”.

Conclusion (17.02.2025): This project only works if the business is fictitious. It wouldn't be desirable for the stakeholders in the real world life. Imagine Bet365 giving major parts of their business to some tiny new bookmaker, who undercuts them by 0,001

8.3.2 Should we be a bookmaker, or only an aggregator?

10.02.2025

- If we are an aggregator showing the "best odds", it would be tough for us to take a margin, as that margin would possibly reduce the odds such that it isn't no longer the best odds, only due to our cut.
- Alternatively, we could do something like a subscriber model for our product. But i suspect that people would quickly calculate that as a percentage cut of their winnings as well, and therefore again reduce the chance that the customer would perceive it as the best odds available. Therefore, we think it would be easier to take a cut, on a service usually not available on other sides, which is acting as a stock broker just for betting. Meaning that people themselves would be able to open a position on some bet, and therefore act as the bookmaker themselves, for other people to buy their odds. This would cost a fee, quite similar to the ordinary fees one pays on the stock market.

Conclusion (04.03.2025): We are mostly an aggregator. We don't act as our own bookmaker, BUT we do allow our customers to act as individual bookmakers through our site, known as “community bets”.

8.3.3 Does this concept give a total increase in new customers across betting sites, or does it eat from the existing customer base.

10.02.2025

- So one of the biggest podcasts our there from Conan O' Brian, who is mainly on Spotify with audio only. They are now experimenting with putting episodes on Youtube with video as well. But they are afraid that it won't give more customers, instead people will migrate from Spotify to Youtube. This is a concern for the betting vendors as well, where there are mainly two scenarios:

- Best case: Some customer who only uses X, instead now uses our platform and therefore indirectly ends up using A, B, C, D.... who never had a chance before to be used by this customer. They all see an increase, because this customer who now have more options and advantages starts betting 250 % more. So even though site X will see a decrease from this customer, there might be other customers who only uses A, that will now also start using X.
- Worst case: Some customer who only bets 50 kr. each month on X, will now still only place 50 kr each month, but now on A, B, X. Thus no additional revenue would be added to the pot.

Conclusion (17.02.2025)**:** We realized this case has too be fictitious, as it doesn't make real-world sense. For example, there is no way Bet365 is going to give their huge market share, to some little unknown bookmaker, that undercuts them with 0,001% - As such, these business decisions doesn't always have to make sense, as the SOLE reason we made this project, was because we found it to be architectural interesting, and not as a real feasible interesting company.

8.3.4 Should our concept only serve sport?

10.02.2025

Most of these sites also have:

- Bingo
- Casino & Live casino
- Poker & other card games
- Raffles
- eSport (One might claim they are the same)
- Quizzes and giveaways

While these might make sense in our business scope, it will be too much for this 1 year project, therefore we only focus on sport - not because it is the best choice, but it is feasible within our scope of this course.

But alternatively, we can implement casino easily through casino providers. So ALL of the betting sites we checked with danish license (42), all used external casino vendors, such as pragmatic.com As such, we can easily an efficiently have a casino page as well, using external vendors. There is no "down-payment", as we act merely as affiliate, and get a provision from the vendors. This might be of conflict for our own betting site key partners, as if they start using our site, the potential income generated on slot-machines, will be reduced from our partners, as we eventually eat their customers. Alternatively is we don't provide casino (slots, live roulette etc.), as to please our key partners.

Conclusion (17.02.2025): We limit ourselves to sports bets for this project.

8.3.5 Special offers & bonuses

10.02.2025

Is it feasible for our site to have offers, such as a welcome bonus?

This might be rough, because that would give the customers multiple stacked welcome bonuses, one from our site, as well as one from the site they are placing the bet on.

Or they might have to be designed in a smart way, we currently can’t think of.

Conclusion (07.03.2025): No, we shouldn't try hard to become our own bookmaker, with too many of our own bonuses. We should mostly be an aggregator for other bookmakers, and let them come with their own bonuses.

8.3.6 Follow the best users, opt-in, always on, or never show?
10.02.2025

Should we allow people to see the most successful users so they can mirror their bets? If so, should this feature always be on, or something the user opts for.

Conclusion (17.02.2025): This would make the project out of scope – we are not going to focus on these parts through diagrams and analysis. This is mostly interesting in the future, after all base features exists.

8.3.7 We must show betting limitations
10.02.2025

The UI must show the limitations of bets. "This bet gives odds 1.8 but max bet amount is 75kr."

Conclusion (17.02.2025): Makes sense, as it is a safety requirement that people can't use money that doesn't exist. We have added to the account overview UI mock-up columns for bonuses.

8.3.8 Legal concerns
10.02.2025

There must be compliance of "Rofus" and "Stop Spillet". Furthermore, there must be a timer that makes the user aware of their time spent betting. All of these and additional requirements from Spillemyndighederne must be implemented.

Conclusion (01.03.2025): As also seen through our QAW, then Spillemyndigheden is a stakeholder, where we MUST abide. For this project, it mostly means adding it in the UI mock-ups.

8.3.9 Separate withdraw-able cash and bonus
10.02.2025

Bonus money staying in the betting site acts differently than cash that can be withdrawn directly. The site must display how much is bonus and how much is not.

Bonuses must remain on the bookmaker, and can’t be transferred to our site.

Conclusion (17.02.2025): Added to UI Mock-up.

8.3.10 Forum
10.02.2025

Having a forum on the site will act as marketing when users encourage each other to place bets.

Conclusion (17.02.2025): Out of scope for this project.

8.3.11 Streaming
10.02.2025

We could use streaming through for example bet365 so users who want to watch matches live can be satisfied, without having to invest in our own streaming setup with people filming the match.

Conclusion (17.02.2025): Out of scope for this project.

8.3.12 BetBørs is apparently already a thing
10.02.2025

Our “community bets” idea already exist on BetFair. We thought it was a unique idea. Even though it already exists, we are going to continue with the idea, as we think the idea is great for showcasing various software architecture concepts.

Conclusion (01.03.2025): Added to UI mock-ups.

8.3.13 International
10.02.2025

Could this be expanded to be used outside of Denmark? Could this site easily be operated in USA as well?

Conclusion (03.04.2025): Out of scope for this project.

8.3.14 Two customer profiles
10.02.2025

We actually have at least two customer segments.

- Users who want to bet.
- Bookmakers. Imagine a newly created bookmaker, who wants to make a splash and quickly attract new users. They could sign up through us, and “out-bid” other vendors with odds, to quickly reach a massive audience.

Conclusion (17.02.2025): Shown as a stakeholder in the QAW

8.3.15 EU wallet
10.02.2025

Should this site serve EU customers, through the newly EU created product “EU Wallet” – This means that we should alternatively mark our diagrams as “MitID / EU Wallet”.

Conclusion (17.02.2025): We just follow what our stakeholder Spillemyndigheden demands. But this is not shown in any diagrams, except the wording “MitId”, which in the future could be updated to “EU Wallet”

8.3.16 Where is the money
10.02.2025

Having them place money on our site makes it easier for the user to just put 200 and then bet where ever, instead of having to place 200 on multiple site. The down side is that we now need to handle security and all the behind the scenes transactions. But this idea proposal, is one of the key differences between just “googling the best odds” and having a central betting platform, taking care of various quirks, such as minimal down-payment for the customers.

Conclusion (04.03.2025): We included a bank to our architecture so we can hold the money, this way the user does not need to deposit money on every site.

8.3.17 Cancel bets

10.02.2025

Users should be able to “cash out”, cancel their bet, etc.

This system should work very similar to placing bets, such as getting rejected and prompted in case they are trying to cash-out and the odds have changed during the process.

Conclusion (04.03.2025): The microkernel can forward cancel requests as well as bet placement requests.

8.3.18 Diagram showing bounds of our own bank connection and various payment providers

17.02.2025

It can be a bit confusing, when and where the user is going directly through our banking connection, and when the user is going through a payment provider such as Visa & MasterCard.

Such as, what is the trace of money, when a user:

1. Deposits money
2. Places a bet
    - The bet loses
    - The bet wins.
3. Withdraws money

Conclusion (04.03.2025): Has been added to UI-mockup and informal diagrams, but still needs a great proper UML diagram.

8.3.19 Statistics

17.02.2025

Most sites support statistics, such as RTP on a specific feature that last X days. We should show general statistics, such as RTP, win-rate, etc.

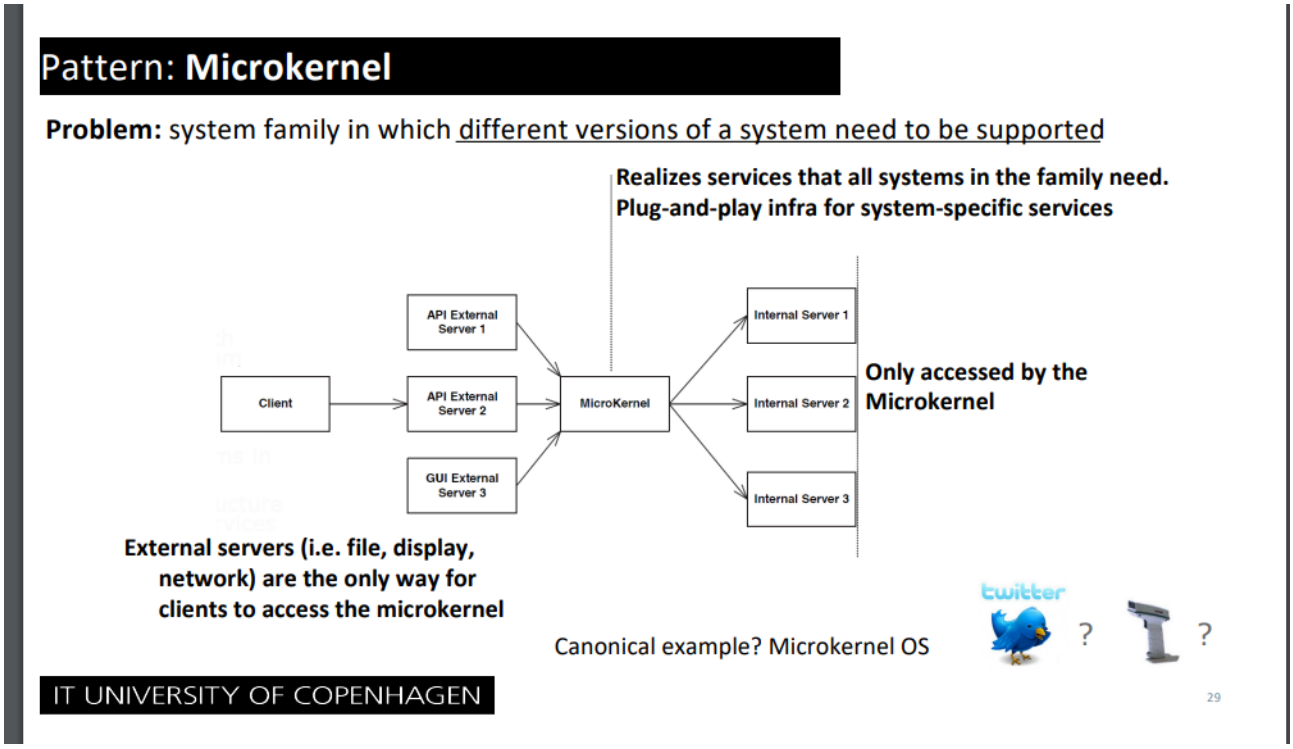
Conclusion (04.03.2025): Out of scope.

8.4 Ideas related to patterns

8.4.1 Pattern for placing bets through many different bookmakers through our site.

10.02.2025

We need a system that handles our external bookmakers different processes, as to minimize the pain of the user, needing to create a profile on multiple sites.

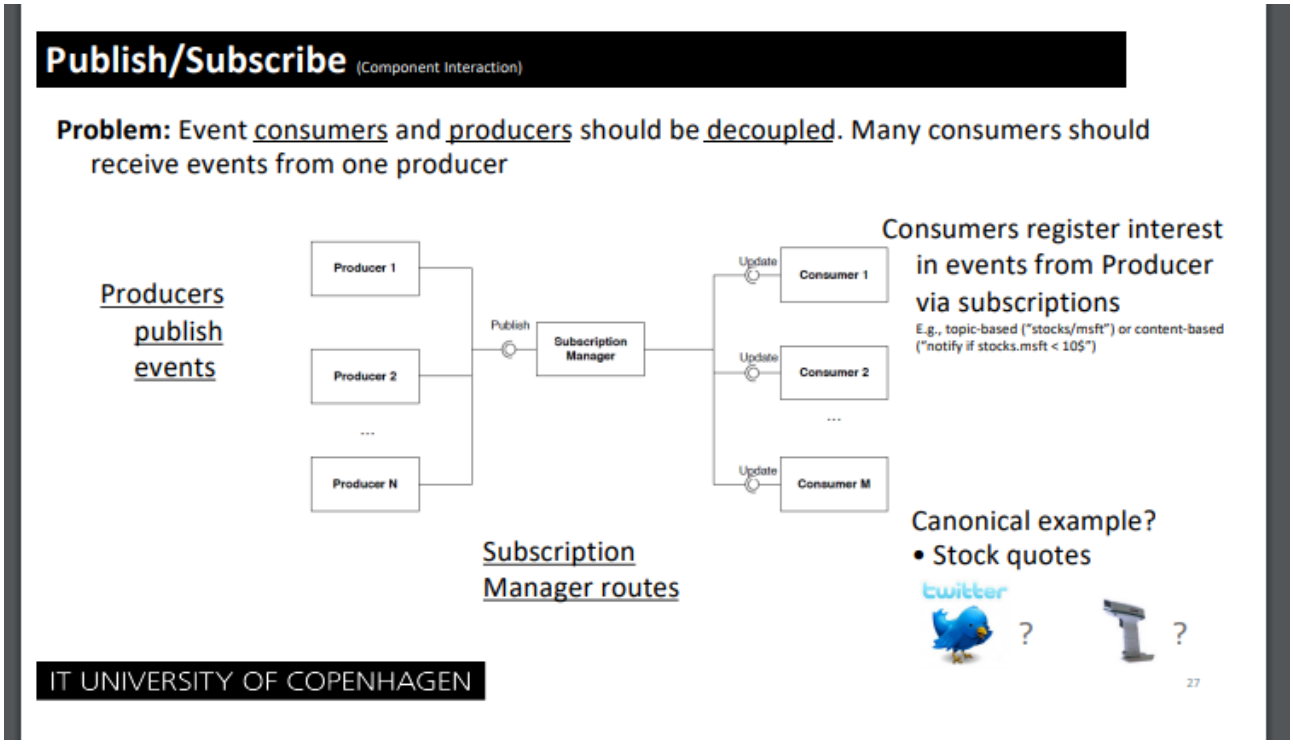


Conclusion (04.03.2025): Implemented in our diagrams.

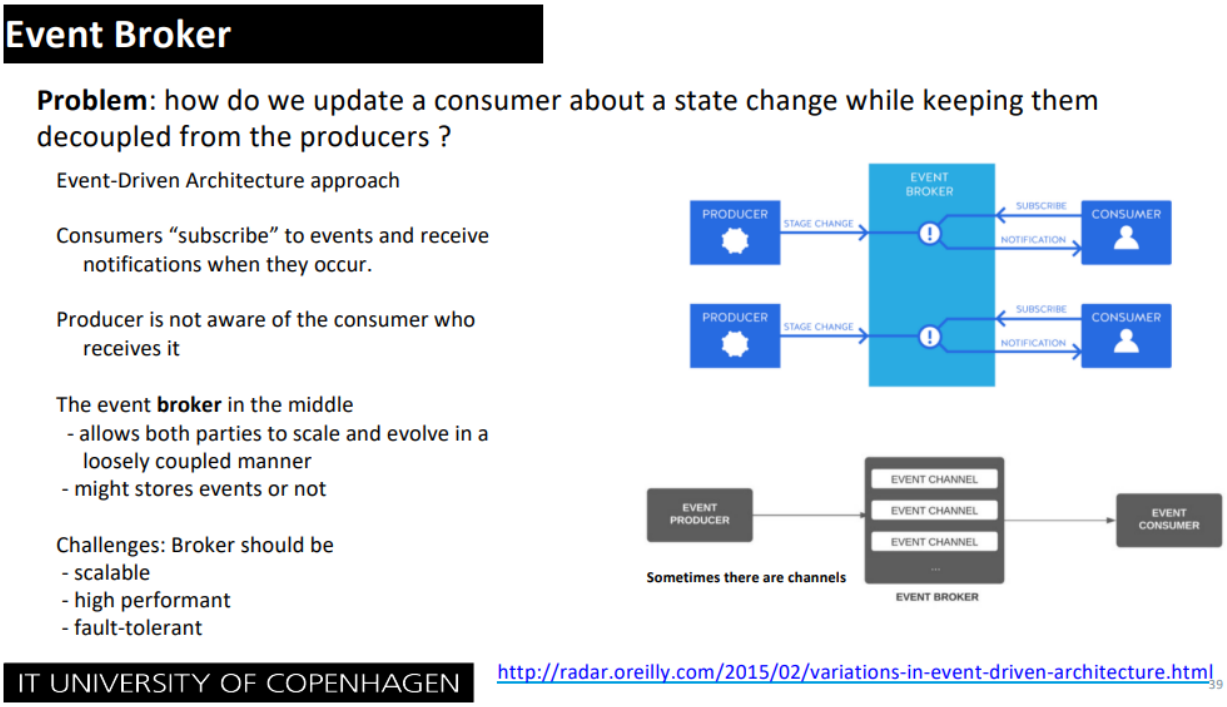
8.4.2 For trigger order like a stock broker, when odds reach X < Y

10.02.2025

If a user wants to place a bet on a match reaching above 2.0 in odds (trigger order), it needs to be done quickly before the odds change. Thus the customer should “subscribe” to a match, where the “publisher” is the odds. If odds reach a certain threshold, the subscriber pipeline should automatically place the relevant bets.



Alternatively, we want to also look at:



Conclusion (03.03.2025): Trigger orders uses Event broker. The producer are “the changed odds on events”, and subscriber are the customers placing trigger orders.
This will then get forwarded to our Microkernel responsible for communicating with the different Actor models using message passing to attempt to place the bet for the customer and handling errors in case the odds have changed, by checking the odds-changed-treshold on the customer, and reattempts if within border, otherwise cancels – acting as a “transaction”.

8.4.3 Database setup - What happens with the database? How many databases? Separate datebases for separate concerns?

03.03.2025

We could envision something like this:

1. Bookmakers update their odds.

2. The updates odds gets sent to our API endpoint, which updates our odds database.

3. The odds database could have a replicated read-only database.

4. Our sports-overview uses only the read-only version.

If so, what do we actually gain from this – we need to prototype this.

So 1 database that takes updates from external bookmakers, and another read-only database for showing general sport odds in the UI? And if so, what should the consistency be? For example, can the odds be 10 minutes behind real-life odds for the UI? What is an acceptable range.

8.4.4 We still need to make every decision regarding the database

06.05.2025

Such as should It be replicated? Which database? What happens when an external bookmaker calls the odds updater endpoint. Do we have multiple databases, such as as database that allows read/write for updating, but then another read-only database that is used when the user wants simple overview of odds for football.

8.4.5 We need to show a diagram for what happens AFTER a placement is made

03.03.2025

Such as updating account information, sending confirmations, etc.

8.4.6 How should the load balancer approach safety and modifiability concerns

05.03.2025

If we choose to use a load balancer instead of an actor model we sacrifice modifiability and safety. How do we ensure that it doesn’t become too much a problem. How should we handle errors and new bookmakers?

Conclusion (10.03.2025): After prototyping, we figured out a combination solution would be good. A load balancer placed in front of an actor model system. The load balancer ensures performance,

whereas the actor model system helps with safety of getting a single bet placed correctly in a proper time.

8.4.7 Compare actor model against alternatives

05.03.2025

Experimental prototyping will reveal more about whether the overhead of actors will be worth the benefits.

Conclusion (10.03.2025): See section about architectural prototyping.

8.5 Ideas related to tactics

8.5.1 Availability

10.02.2025

We need to monitor the health of a lot of different external sites, and should use tactics such as “heartbeat”

Conclusion / What happened (04.03.2025): Added in the tactics section for deliverable 2.

8.5.2 Should we prioritize certain bets?

03.03.2025

Such as, should a bet placement of 1,1 million be prioritized above a freebet of 50 kr.

If so we might want to look at: Tactic: Performance: Control Resource demand: Prioritize events.

Conclusion / What happened (04.03.2025): Added in the tactics section for deliverable 2.

8.6 Discarded decisions

8.6.1 Placing multiple bets at once from different bookmakers with a scenario where one fails

10.02.2025

On ordinary betting sites users can place multiple bets “in a package”, where the site is able to reject the whole package if one bet fails (such as a rapid change in odds). Our system will not be able to handle this, because the package might contain different sites, and as such we can’t cancel a bet on site A, if a bet fails on site B. As such it would be easier to not support package bets on our site.

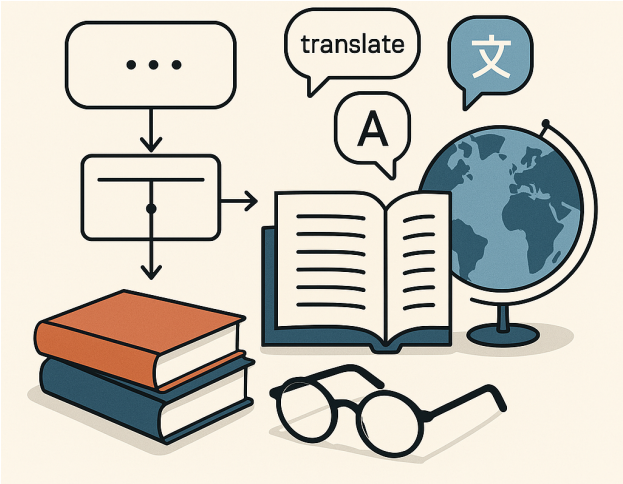
8.6.2 Data-scraping

11.02.2025

We are not going to do data-scraping. If we do data-scraping, it would be hard to actually place bets on the respective sites, without proper confirmations etc. It would also be insanely resource intensive, to data-scrape a bunch of sites and also expect data to be completely fresh for the user. To avoid complications of legal situations and latency, we are going to make an assumption, that all bookmakers are willing to participate and already have an API giving the necessary data.

Conclusion (17.02.2025): This is a fictitious case, where we pretend all the bookmakers have the proper needed API's for this to work.

9 Jacob Grum (Jacg@itu.dk) – Architectural reconstruction of Zeguu



Date:
11/05/2025

Course:
KSSOARC2KU,
Software Architecture, MSc (Spring 2025)

10 Introduction & Methodology

Introduction:
This report presents a detailed architectural reconstruction of Zeeguu-API, an open-source Python backend system. The full system in action can be seen here: https://zeeguu.org/.

Modern software systems evolve continuously, which can lead to deviations between the intended design and actual implementation (architectural drift & architectural erosion). To mitigate these issues, we employ tools such as Radon and Pyan (see sources and full list below), to reconstruct high-level views of the system's architecture. This concise document offers developers and architects diagrams of Zeeguu-API, providing the clarity needed for informed decision-making and potential reengineering efforts.

Data gathering:
I am using as-is-implemented source code from the backend project of Zeeguu: https://github.com/zeeguu/api, analyzed using off-the-shelf tools, to play around, compare and generate various module views.

Knowledge inference:
I want to reconstruct various module-views. Showing their static dependencies and various product metrics such as lines of code, number of functions, cyclomatic complexity & their dependencies, resulting in polymetric views that combines size, complexity, and dependency information to aid developers understand the system and guiding future reengineering efforts.
I will also try to use abstraction techniques such as filtering files, by utilizing product metrics as filter parameters, and removing standard-library dependencies on the graphs.

The tools used were:
Mostly unsuccessful:

- **Architectural Lens**: https://github.com/Perlten/Architectural-Lens/tree/master
- **PyCallGraph:** https://pycallgraph.readthedocs.io/en/master/
- **Snakefood:** https://github.com/blais/snakefood
- **Prospector**: https://prospector.landscape.io/en/master/

Average success:

- **Pylint's Pyreverse:** https://pylint.readthedocs.io/en/latest/additional_tools/pyreverse/index.html
- **Pyan3**: https://github.com/Technologicat/pyan

Mostly successful:

- **Radon**: https://pypi.org/project/radon/
- **GitTruck**: https://github.com/git-truck/git-truck

From now on, I want to use the technique Mircea used during the lecture, by marking certain text in **bold**. If you **only read the bold parts,** you get a **concise understanding of the most important parts**.

11 Results

Structure:

Jacob's Individual Appendix: Shows **code structure** & **visualization setups**, as well as **specially chosen illustrations** that are the **most interesting** and have **hyperlinked references throughout the report**.

Other illustrations created during the process: In repo output\radon\average_xx_for_yy or worst/best_xx_for_yy shows many non-polymetric illustrations, mostly for documenting the path of learning I took. **These illustrations can be safely skipped**

11.1 Radon results

For detailed **polymetric visualization setup and explanation** see **[Radon illustrations & Setup].**

Initially **I started with** simple bar-charts illustrating for **product metrics for the modules and functions**, seen in output\radon\{average\worst\best}_xyz. Here we can quickly conclude **most logic resides inside the Tools and Zeeguu packages**, which are going to be my **main area of attention** from now on. **Figure 26 to 31 shows graphical polymetric views**. Each graph has **various amount of filtering** and is shown with **either Api/Tools or Api/Zeeguu as root.**

I would argue **one of my best outputs with this tool is figure 28 showing an abstracted module view of the Zeeguu package**. Here we can quickly see that much of the **complex work takes place in core.content_retriever.article_downloader**, having both **high cyclomatic complexity (CC)**, **many lines of code** & calling **functions with many parameters**. This might make it an **interesting focus**, if one were to attempt a **refactor with focus** of improving **maintainability** or **modularity quality attributes**.

My **other best output for the Tools package is figure 30**, where we see that **report_genator.generate_report** being a **main entry point**, and most of the logic regarding **updating bookmarks** is semi **high in CC**, and everything regarding **bookmarks is higher in complexity than everything else in Tools.**

Semi-conclusion:
Both figures have the **identified key modules** written in bullet-form beneath the figures. These **views** quickly s**hows where most of the logic takes places**, and how difficult the logic might be to understand, and as such **affects the cognitive load of a developer**. The figures also reveals important **entry-point modules** that handle the **core pipeline logic of specific features**, often making them **great starting points for a developer working on a specific feature**.

I like **Radon** for giving me a quick **simple higher-level module viewpoint**, with **various product metrics** making me able to generate different instance of views.

11.2 Pyan results

I am **exploring tools that provide more insight into the key modules** found in figure 28 & 30, and I attempted to get Pyan to do so, this time **without** looking at **product metrics**, **but all dependencies** this time.

For **detailed Pyan visualization setup** and **explanation** see **[Pyan illustrations & Setup]**

Something **I like** about **Pyan** is its **ability to generate SVG output**s, meaning you can *ctrl+f* in the output. For example it can be hard to locate zeeguu/core/something, in my visual outputs from Radon, but with Pyan I can.

The **entire output is gigantic**, and not really viewable in a report format. But luckily I am **only interested in zooming in on specific modules**. These views would give a new developer a good overview, if they were to work on that specific feature.

Figure 33 to 42 shows some interesting **call-graphs for the identified modules** residing inside the **Zeeguu package**, whereas **figure 43 to 48** shows the identified modules residing inside the **Tools package**. I was **unable to filter out irrelevant crossing arrows,** nor remove them when zooming in – **sorry**.

I would argue my **best outputs** with this tool is **figure 33** for the **Zeeguu package**. And **figure 43** for the **Tools package**. Both show some of the **most complex modules in the system** and **their call-graph**, making it easier for a new person to be able to understand and work with these modules.

Semi-conclusion:

Both my tools, **Radon & Pyan worked together**. **Radon** was great at providing **filtered module views**, **with only key-nodes** shown **with** the addition of **product metrics**, but **lacking in dependencies**. **Pyan** was great at getting a more **in-depth view** of the found key-nodes, showing the full non-abstracted **static call-graph**, and therefore provides **more information regarding dependencies across modules than Radon**, with addition of showing other **modules calling each other**, whereas **Radon shows which functions calls each other**.

11.3 GitTruck results

Due to the size this report already is – I am going to **focus on only the Zeeguu Package for GitTruck** and not on the Tools package. I found **GitTruck** helpful for getting **insight into the software evolution**, such as finding **modules that often change, and by whom.**

I would argue that the **best architectural key-information** extracted **from GitTruck**, for a new developer is information about **who to ask for help**, and a **fast interactive package overview**.

Churn - Files receiving the most commits: See figure 49

Time since file was changed: See figure 50

An idea I have here, is that I could **pair this up with maintainability index**. Such as:

- This file **changes** all the time + has **bad maintainability** index = we should **refactor** this.
- This file almost **never changes** + **bad maintainability** index = Maybe we should **delay refactoring this** – “if it ain't broken, don't fix it”

Top contributors & Code-ownership: See figures 51 to 53
Over time Mircea was replaced as the top contributor by Tiago, while still being involved.

Truck-death number / Files with only 1 author: See figure 54
As the saying goes, if we kill 2 ~~birds~~ developers with 1 ~~stone~~ truck, it would be devastating for ZeeGuu. This project is **mostly maintained by Mircea Filip Lungu & Tiago Ribeiro**

In summary
GitTruck provides great **evolution insight** with great **overview of contributors** and a**ctivity in files**, GitTruck also provides a great **interactive package view**, that I failed to generate with Radon and Pyan, thus making **this trio of off-the-shelf tools able to cover a wide range of problems**, making them **greatly complement each other**.

12 Discussion

12.1 Limitations of my approach

- **Hard to make images fit nicely** and being fully readable in the report. In reality you would like to inspect the individual image files in Output/{toolname}/{image}.png, allowing to zoom and pan easily.
- **Hard to ensure no overlapping labels**, making some text/arrows unreadable.
- I often wanted to use my tool **without** having to deal with the **dependencies from the testing package** and often couldn't figure out the correct filtering flags, and therefore I sometimes **removed the testing package**.
- **Pyan did not show all dependencies**. Example endpoints only showed their top function, but no dependencies.

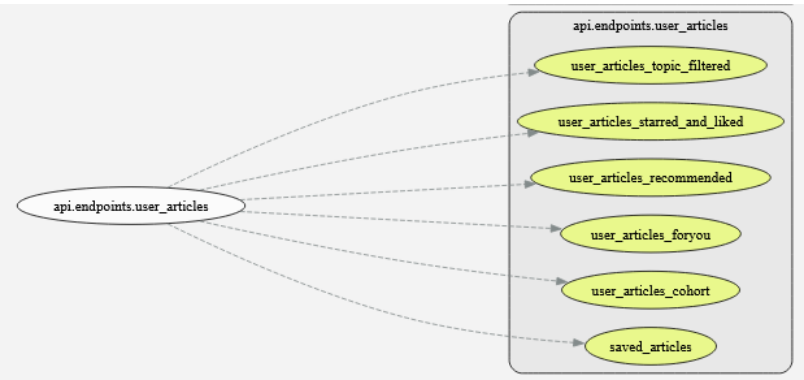


Figure 24: Example of lacking dependencies

As seen in 24, no further dependencies is shown from this endpoint, even though not all logic is shown here. That means, we don't potentially know how far this error reaches and what other information we are lacking.

- I **wanted** my tools **Radon** and **Pyan** to show **more information about the packages**, but was unable to do so.

12.2 If I had more time I would...

- Look into **commit messages**, and the information they provide.
- More **in-depth with fewer tools** – instead of mostly general information from many tools.
- Inclusion of a dynamic viewpoint
- **Better visualization with no overlap** at all, and **fitting everything in a readable format** for an A4 paper.
- Choose a **sub-level deeper**, such as instead of using Zeeguu/ We could use Zeeguu/core.content_retriever**,** in other words **try a different abstraction level**. **I might have gone too high-level abstraction.**
- At some point after creating the whole Pyan analysis, I went back and updated the filters to my Radon illustrations, but **did not have time to make a whole new complete Pyan analysis**. As such there might be a **small discrepancy between figure image**, the **bullet point list of key-modules** and the **zoomed in version through Pyan.**

13 Jacob’s Individual Appendix

13.1 Description of code, link and structure

- **Repo:** https://github.com/Grumlebob/ArchitecturalRecovery
- **Main code:**https://github.com/Grumlebob/ArchitecturalRecovery/blob/main/Tools/Jacob.ipynb

Structure:

```
Data
 └── api <- Root of solution i am analyzing lives here. Cloned 04.04.2025
     └── tools
     └── zeeguu
         ├── api
         │   └── endpoints
         ├── core
         ├── config
         └── logging
Tools
 └── toolscript.ipynb <- My scripts lives here.
Output
 └──radon
     └── someGraph.png <- Image output of tools should be here
```

Time allocation: Initially I played around with each tool in their own tool.ipynb for two whole days, while exploring new tools to try. Afterwards I began taking the interesting parts into a single python notebook, by selecting specific tools to go in-depth with. Then I began analyzing and writing for next 3 days, while revisiting various illustrations and trying to improve them visually.

13.2 Radon illustrations & Setup

I used **product metrics** as an **abstraction**, by getting the following:

- **Lines of Code(LOC)** and **number of functions (NOF)**
- **Cyclomatic Complexity (CC)**, which looks at number of decisions a block of code contains, as well as their **CC rank**[1]
- **Maintainability index (MI).** Which is a **combination of 4 other metrics**[2], one being the Halstead index[3], which looks at distinct number of operators and operands compared to the total. As well as the CC, Source Lines of Code (SLOC), and number of comments.
  A low Maintainability index is bad, and a **high one is good**.

I Also utilized **abstraction** by adding **filters**:

- **For modules:**
  - Minimum **functions per module**
  - Minimum **average CC threshold**
  - **Take top X modules** based on above **CC metric**.
- **For functions:**
  - Minimum **CC per function**
  - **Take top X functions** based on above **CC metric**.

For **visualization**

- Using Graphviz to **reduce overlapping** node text
- **Labels** and **Title** for **explanation**, with a **Legend** for **mapping color and shapes to logic**.
- **Root node** = **Zeeguu** or **Tools** package.
- **Module node** = a **python file**.
- **Function node** = a **function in a python file**.
- **Size of node**: **Lines of Code** (Min = 40 % size, and Max = 350 % size),
  - For **functions** I used radon raw -j, to give **LOC per function**.
- **Arrows & Thickness of arrows**:
  - For root → modules, it is **number of calls** (how many times a module is used).
  - For modules → functions, it is **parameter count**, often indicating higher coupling, such as a function nearly only used by one module, but needs such a specific setup, that the function is highly coupled to the module.

Resulting in the following legend seen in figure 25 for all Radon illustrations.

Each figure will have its own page, for better readability. But for maximum readability, go to repo and find output\radon\api_zeeguu_xyz

---

1 https://radon.readthedocs.io/en/latest/api.html#module-radon.complexity

2 https://radon.readthedocs.io/en/latest/intro.html#maintainability-index

3 https://radon.readthedocs.io/en/latest/intro.html#halstead-metrics

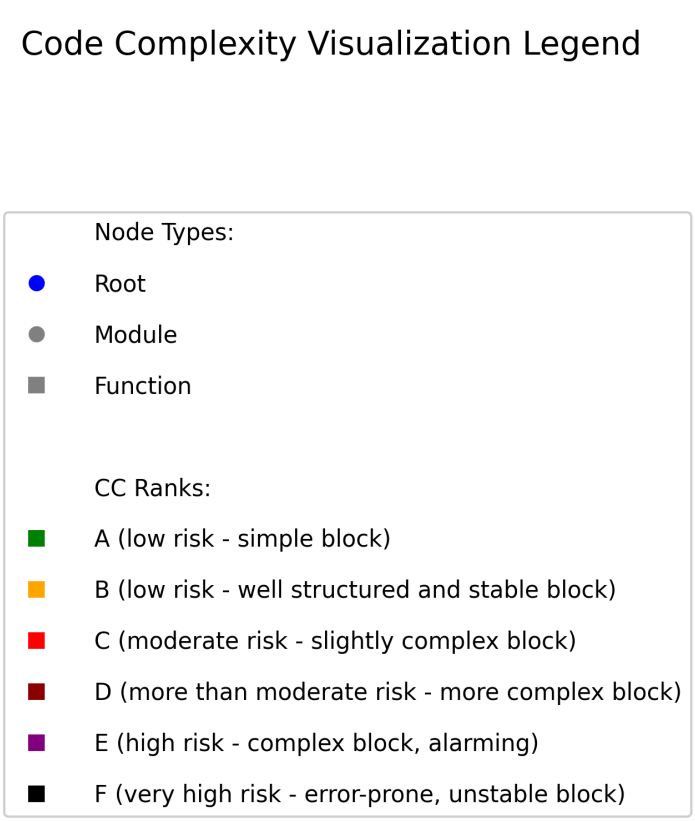


Figure 25: legend for all Radon illustrations

Showing /Api/Zeeguu as root:

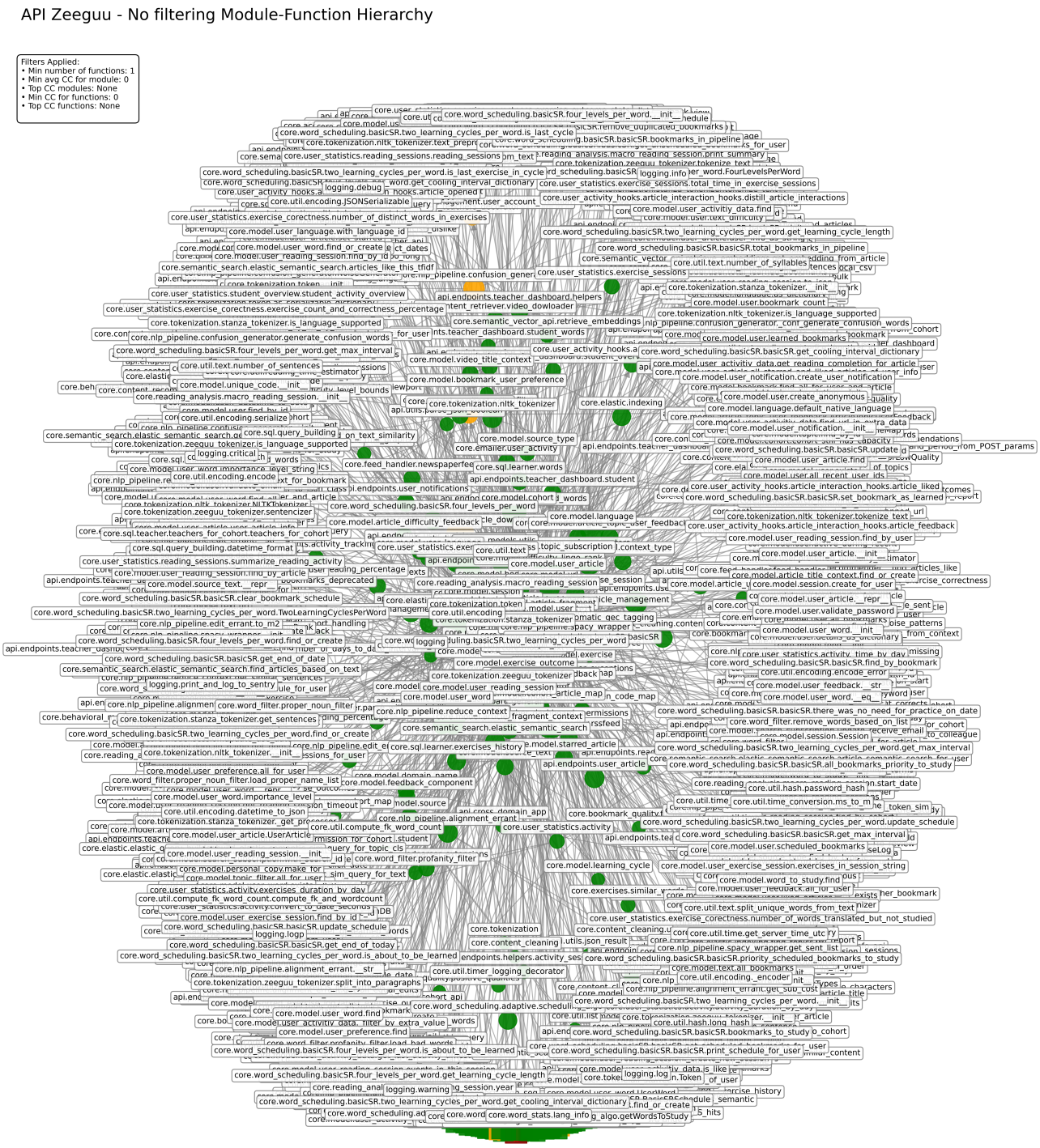


Figure 26: Module - function hierarchy for Zeeguu project, with NO filtering - resulting in a big ball of spaghetti

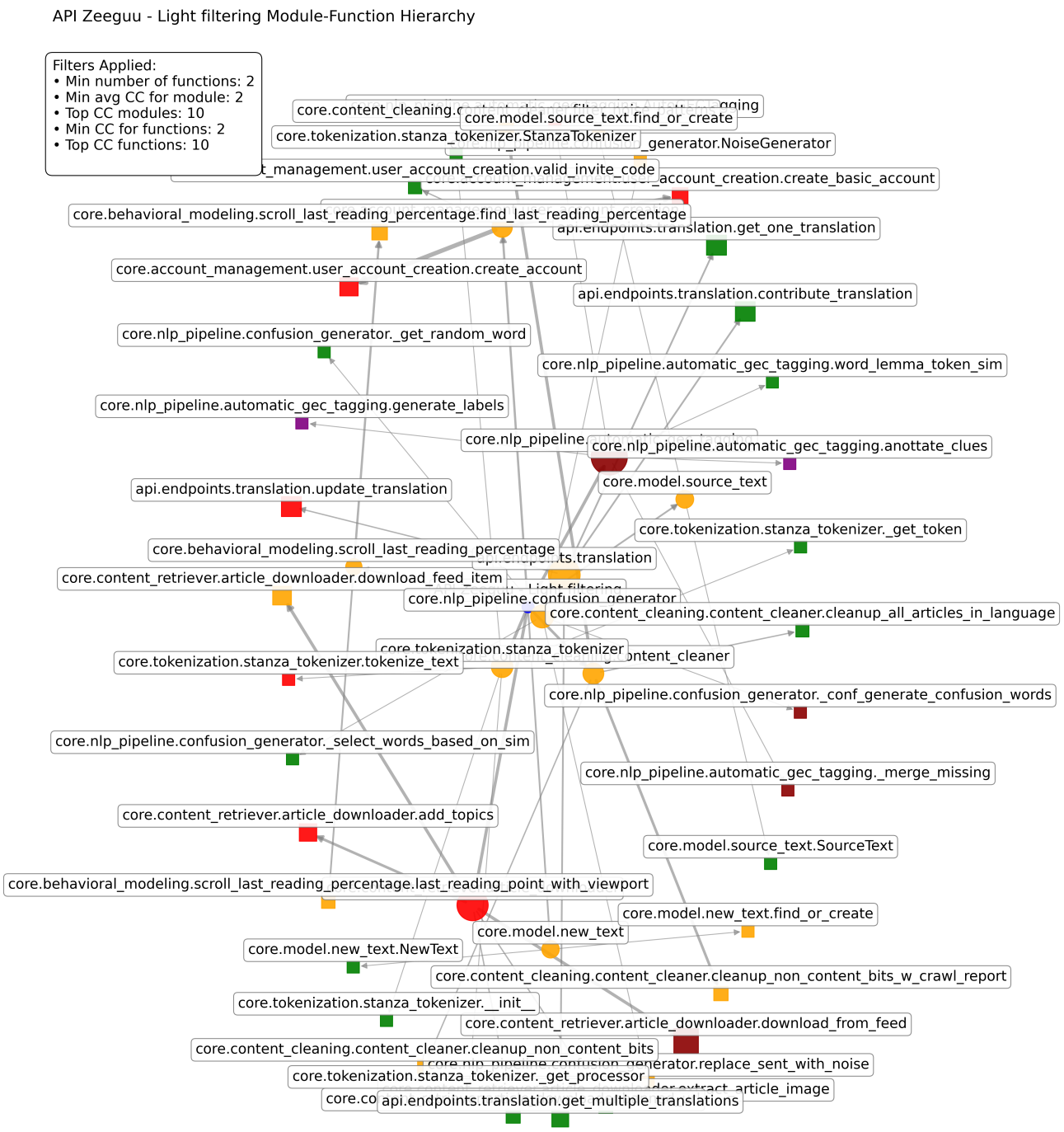


Figure 27: Module - function hierarchy for Zeeguu project, with SOME filtering

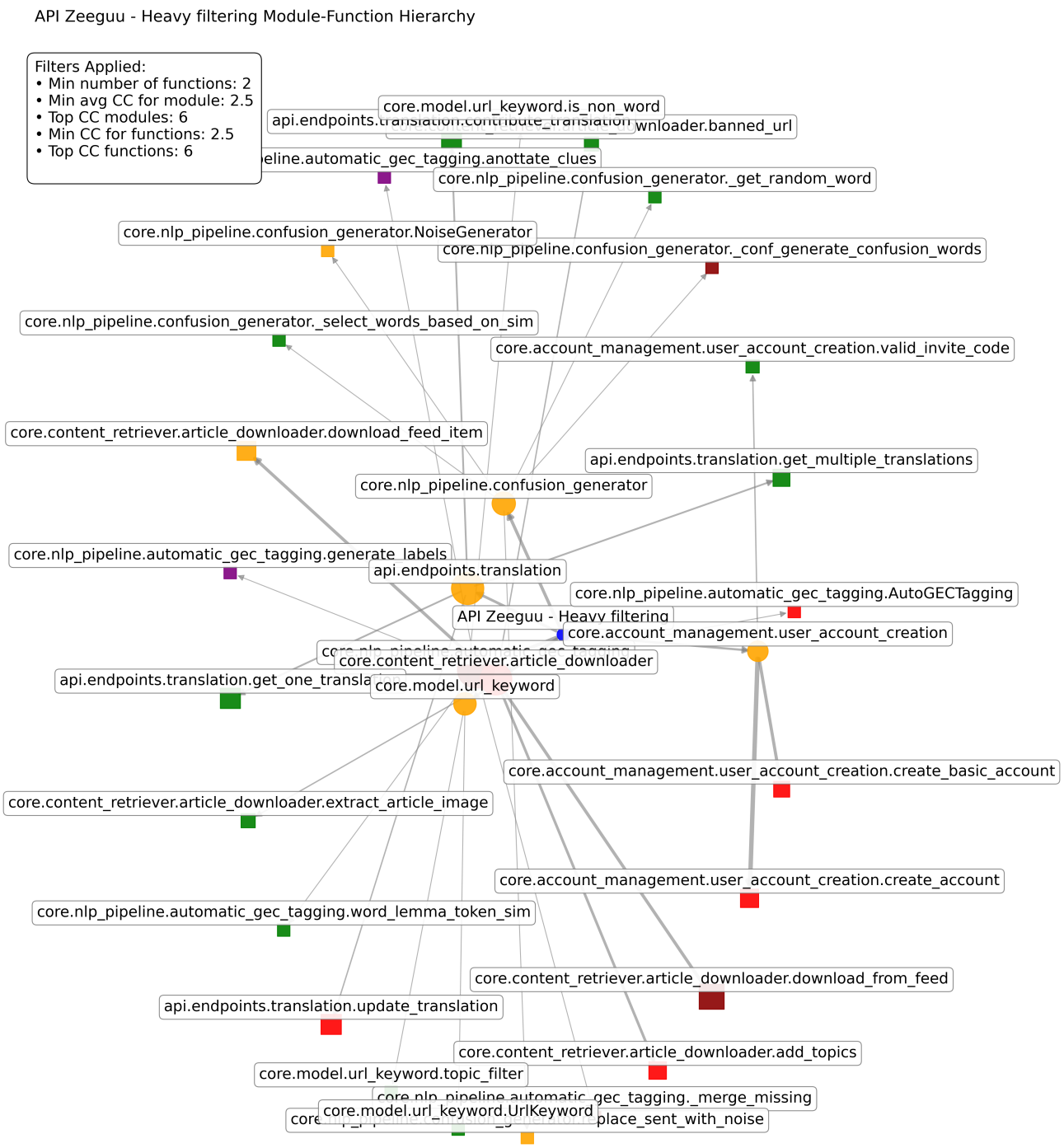


Figure 28: Module - function hierarchy for Zeeguu project, with HEAVY filtering

Some important modules/functions from fig 28 written in text:

- core.model.user_activity_data
- core.nlp_pipeline_automatic_gec_tagging
- core.tokenization.stanza_tokenizer
- core.content_retriever.article_downloader
- core.account_management.user_account_creation
- api.endpoints.translation
- core.nlp_pipeline.confusion_generator

Showing Api/Tools as root:

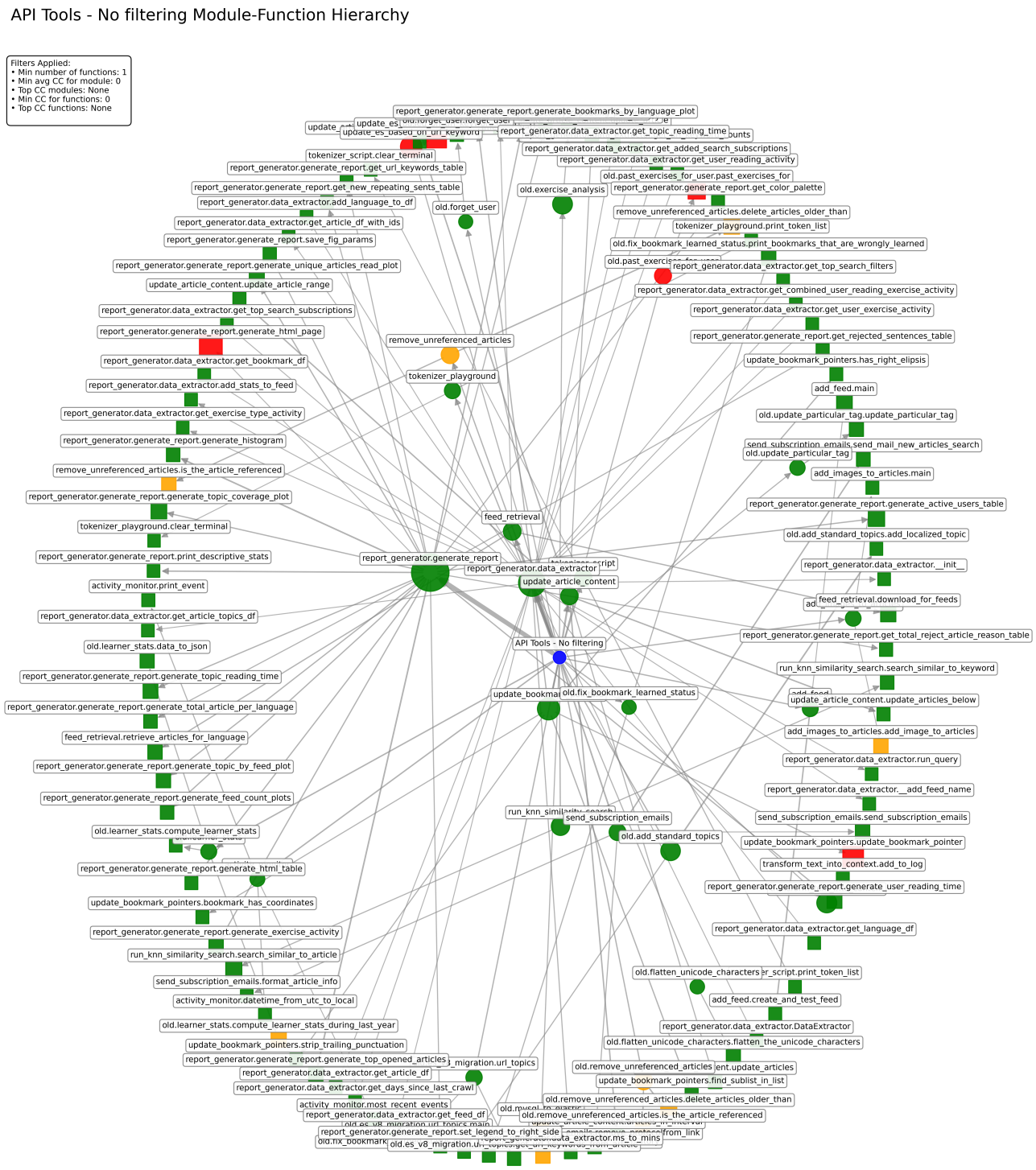


Figure 29: Module - function hierarchy for Tools project, with NO filtering

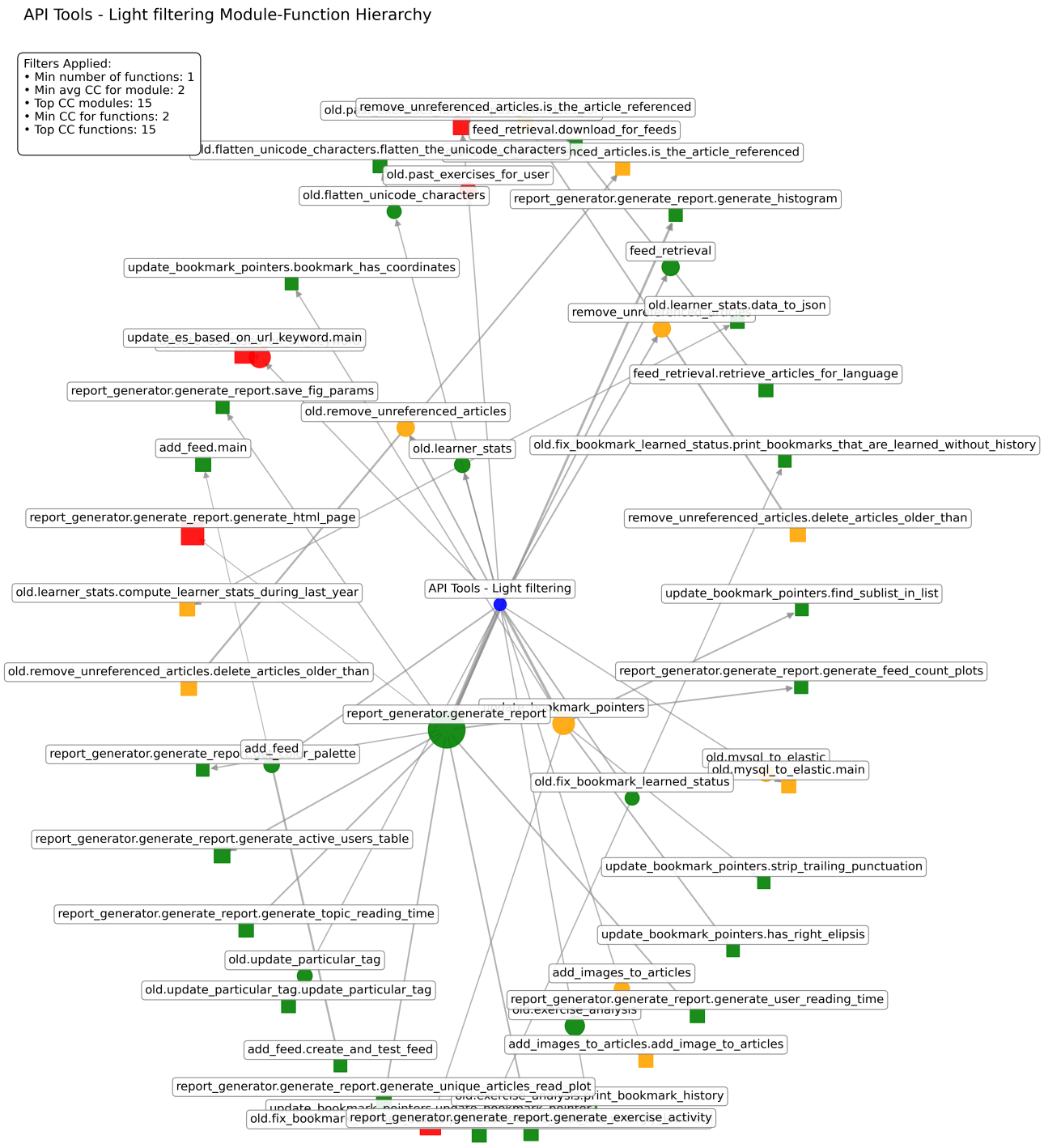


Figure 30: Module - function hierarchy for Tools project, with LIGHT filtering

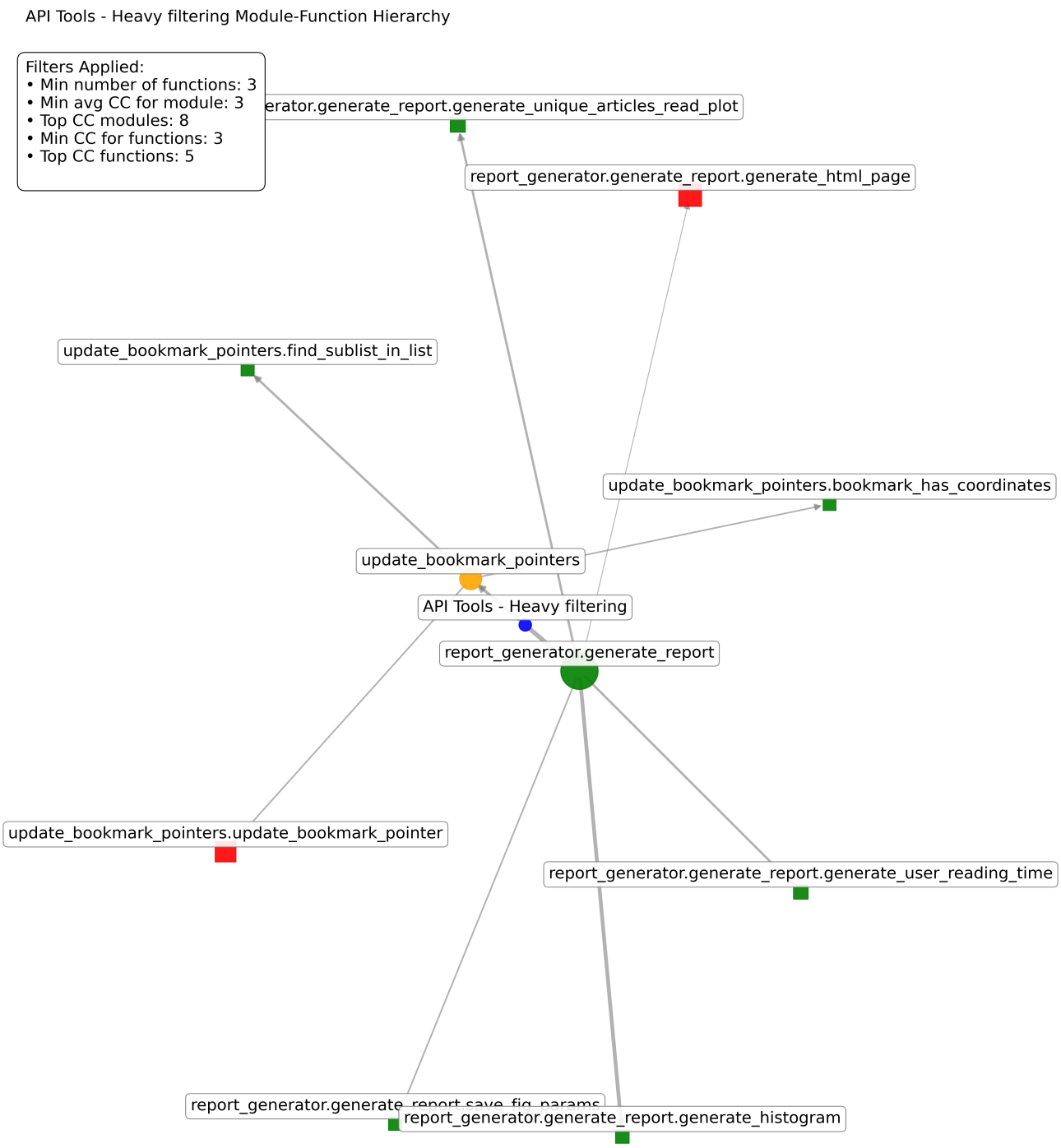


Figure 31: Module - function hierarchy for Tools project, with HEAVY filtering

Some important modules/functions from fig 31 written in text:

- report_generator.generate_report
- update_bookmark_pointers
- old.mysql_to_elastic
- remove_unreferenced_articles
- old.past_exercises_for_users

13.3 Pyan illustrations & Setup

Command used:

pyan3 *.py */*.py */*/*.py */*/*/*.py -d -G -g -c --dot-rankdir LR --svg > ../../../output/pyan/JacobPyan{Tools/Zeeguu}

Which makes it colored, grouped by module, looks at a depth of 3 from ZeeGuu as root, oriented left to right, and outputs in svg format.

Key elements explained:

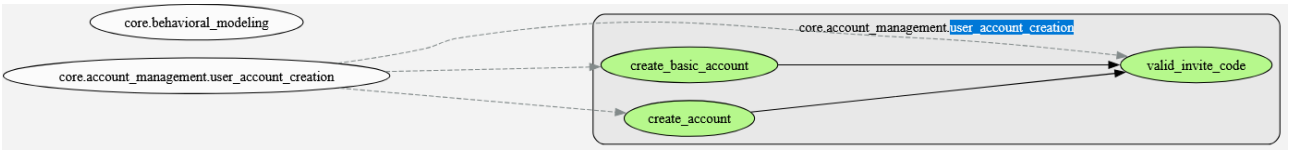


Figure 32: Example image, with various elements

White Nodes: Modules (python files)
Colored Nodes: Functions (A function in a module)
Dashed arrow: This function - - > lives in this module (Defined-in)
Solid arrow: This function → Calls that function
Grey box: These functions [] belongs to this single module.

Some important modules from fig 16:

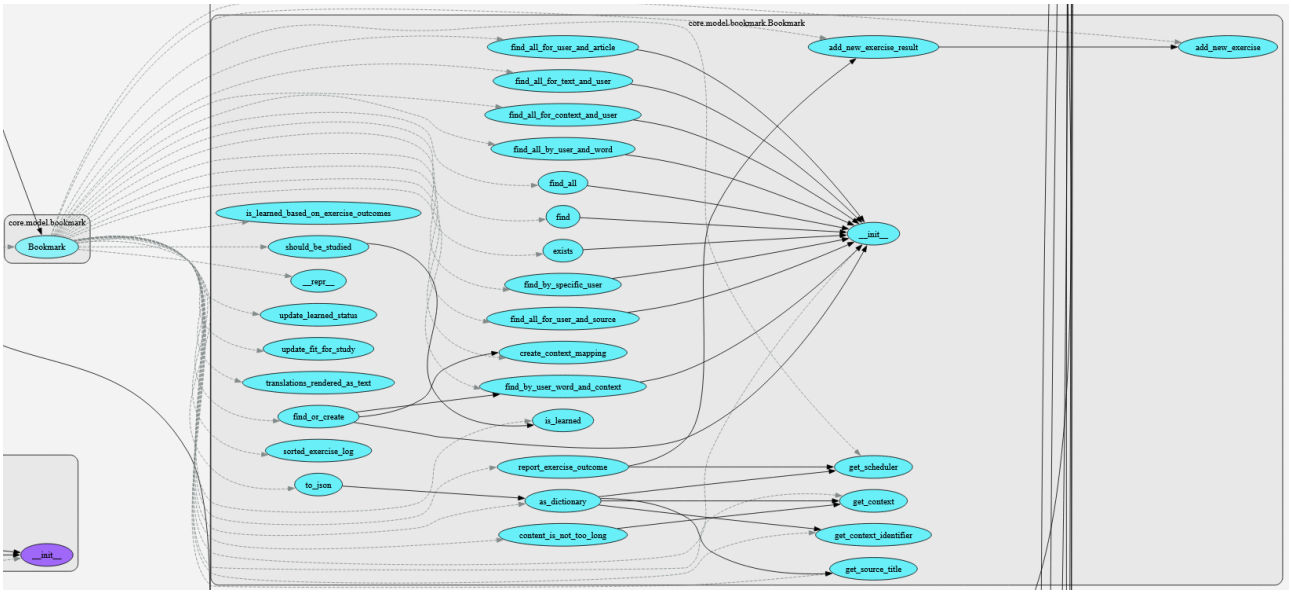


Figure 33: Zoomed in view of core.model.bookmark

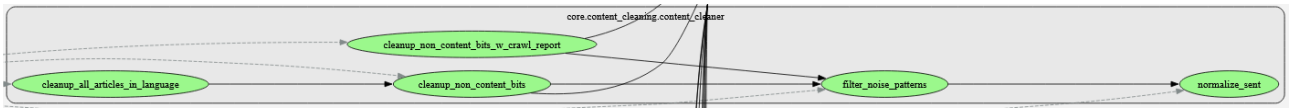


Figure 34: Zoomed in view of core.content_cleaning.content_cleaner

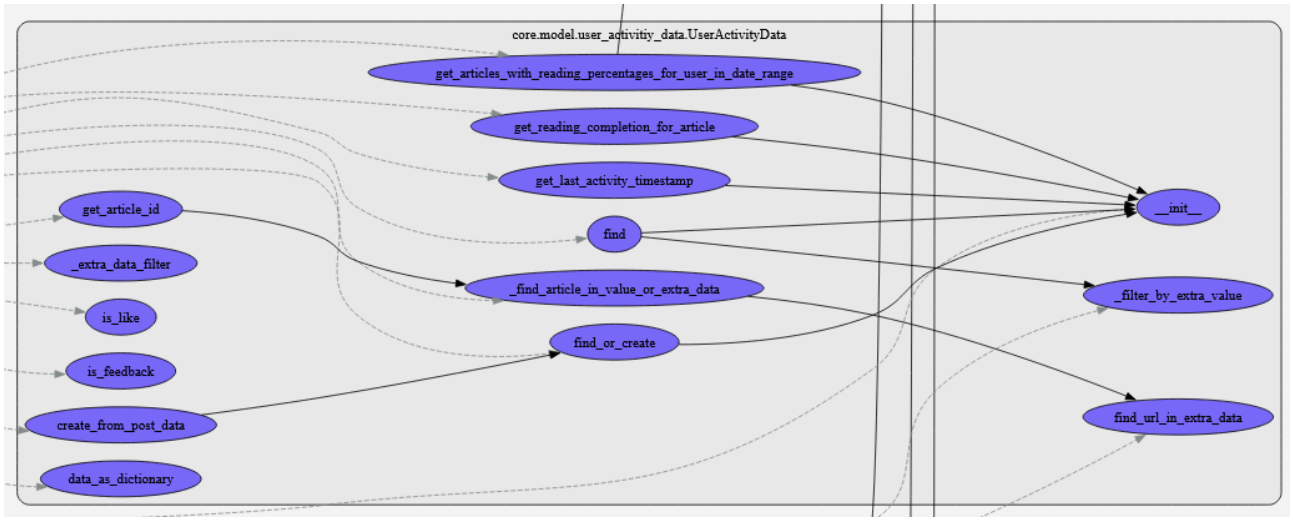


Figure 35: Zoomed in view of core.model.user_activity_data

With the usage to core.model.user_preference.UserPreference.Set outside this image.

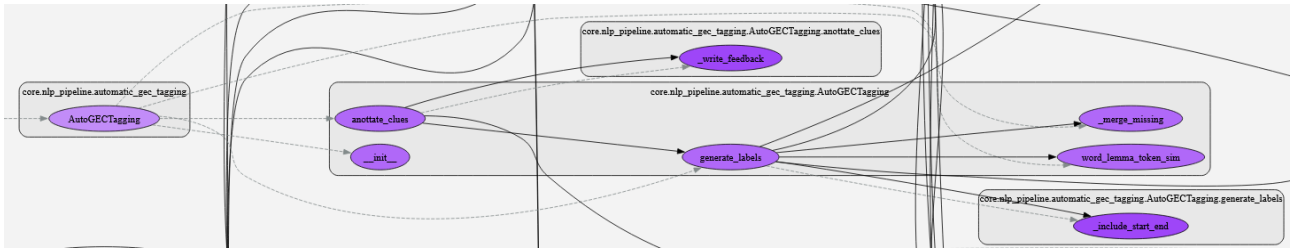


Figure 36: Zoomed in view of automatic_gec_tagging

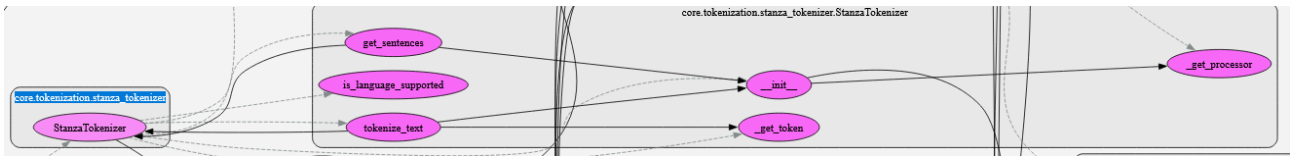


Figure 37: Zoomed in view of core.tokenization.stanza_tokenizer

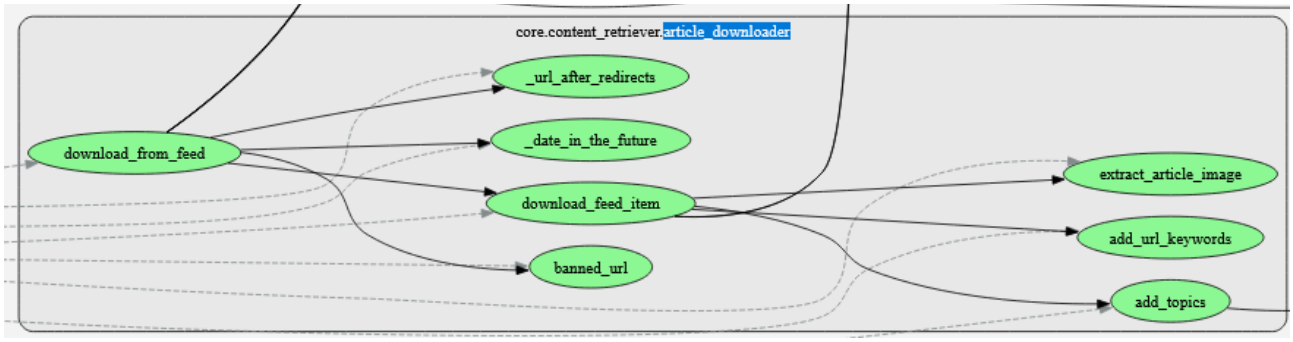


Figure 38: Zoomed in view of core.content_retriever.article_downloader

With usage to core.model.user_preference.UserPreference.Set & core.content_retriever.crawler_exceptions outside this image.

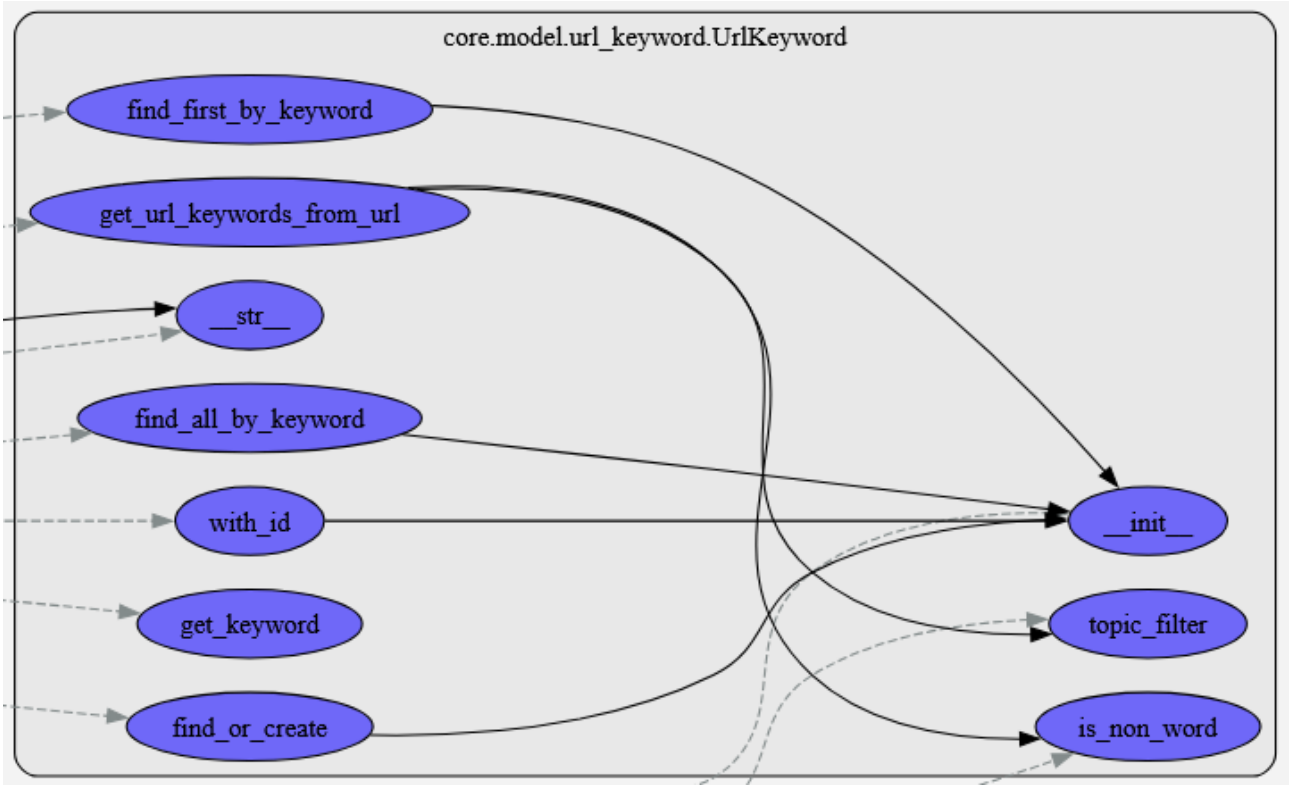


Figure 39: Zoomed in view of core.model.url_keyword

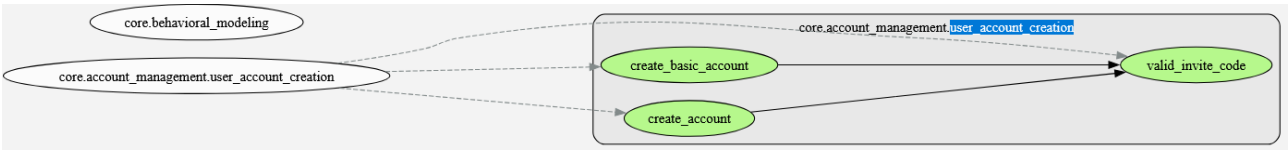


Figure 40: Zoomed in view of core.account_management.user_account_creation

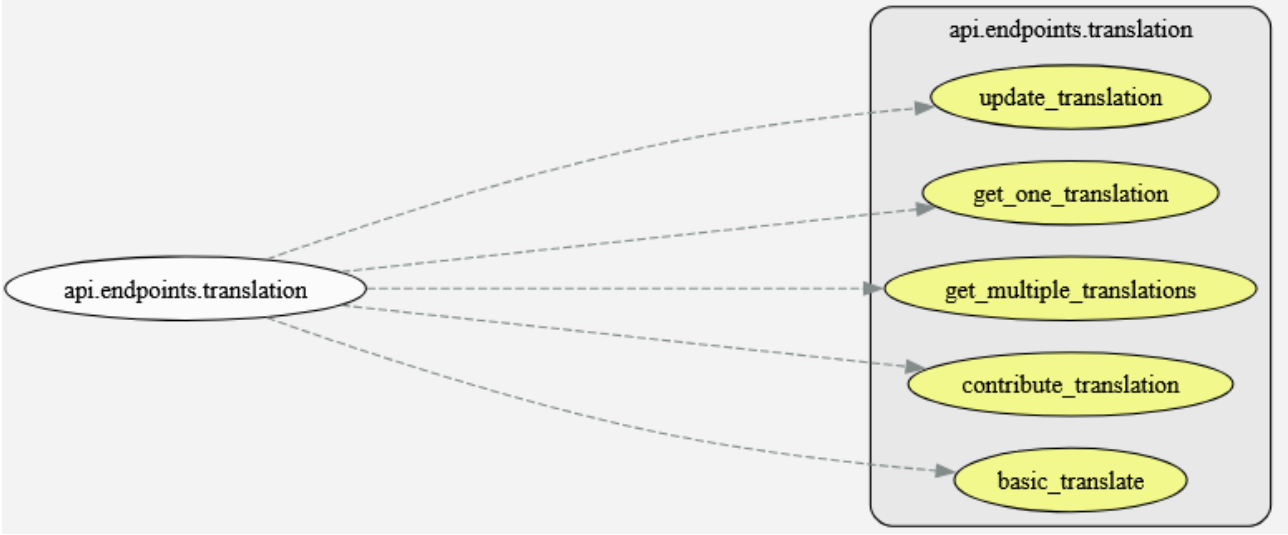


Figure 41: Zoomed in view of api.endpoints.translation

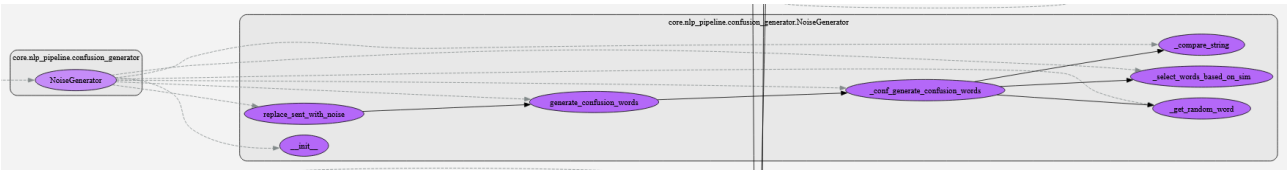


Figure 42: Zoomed in view of core.nlp_pipeline.confusion_generator

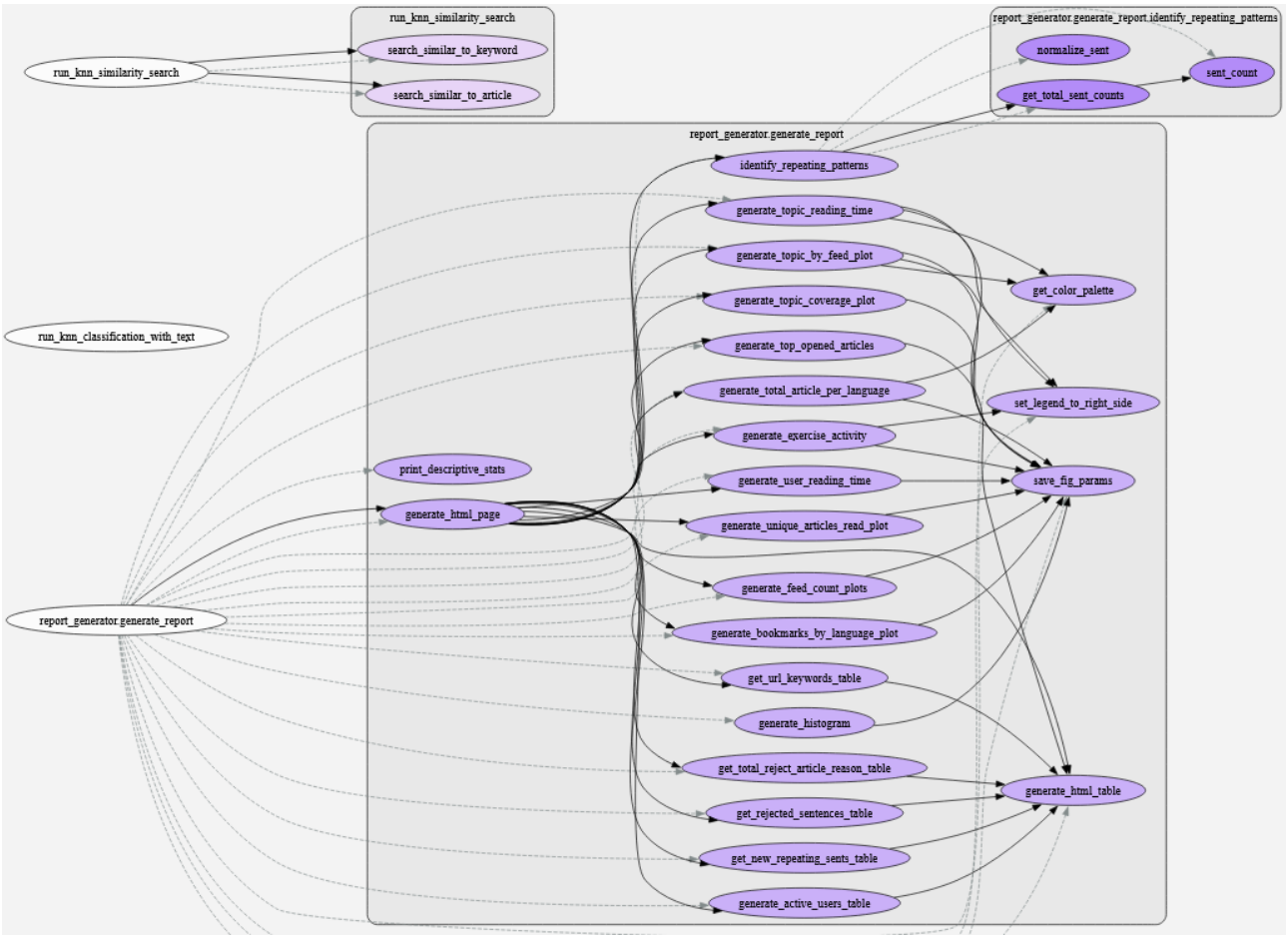


Figure 43: Zoomed in view of report_generator.generate_report

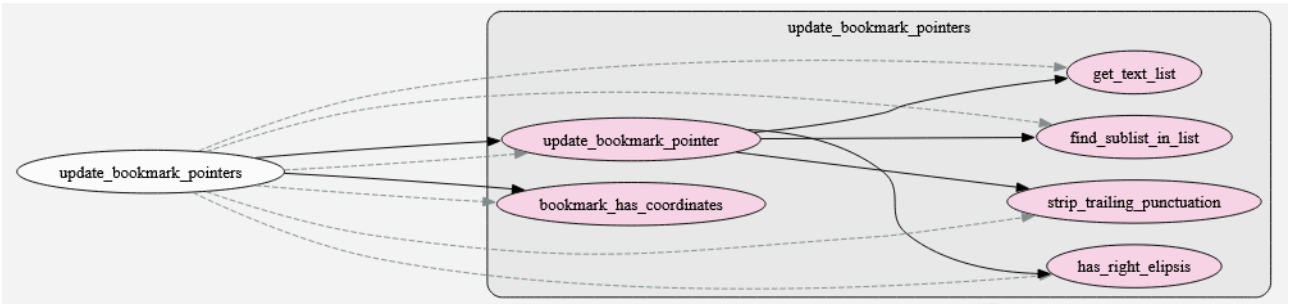


Figure 44: Zoomed in view of update_bookmark_pointers

Figure 45: Zoomed in view of old.mysql_to_elastic

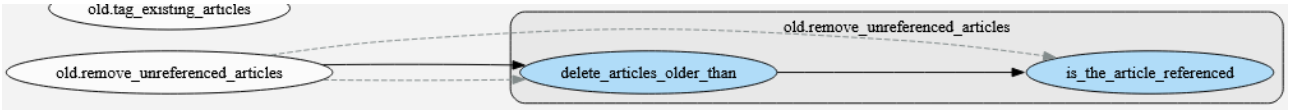


Figure 46: Zoomed in view of remove_unreferenced_articles

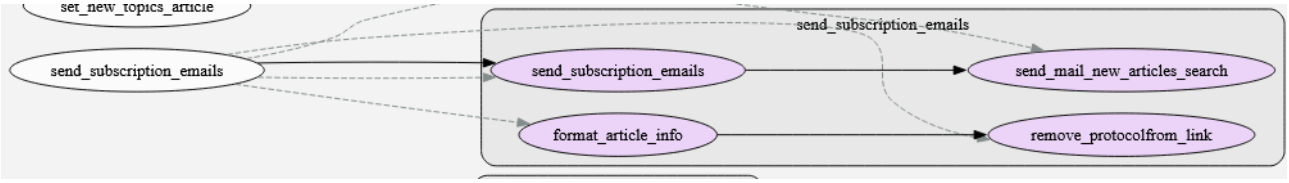


Figure 47: Zoomed in view of send_subscription_emails

Figure 48: Zoomed in view of old.past_exercises_for_users

13.4 GitTruck illustrations

https://github.com/git-truck/git-truck

Command used from where .git folder resides:

npx -y git-truck

Churn - Files receiving the most commits:

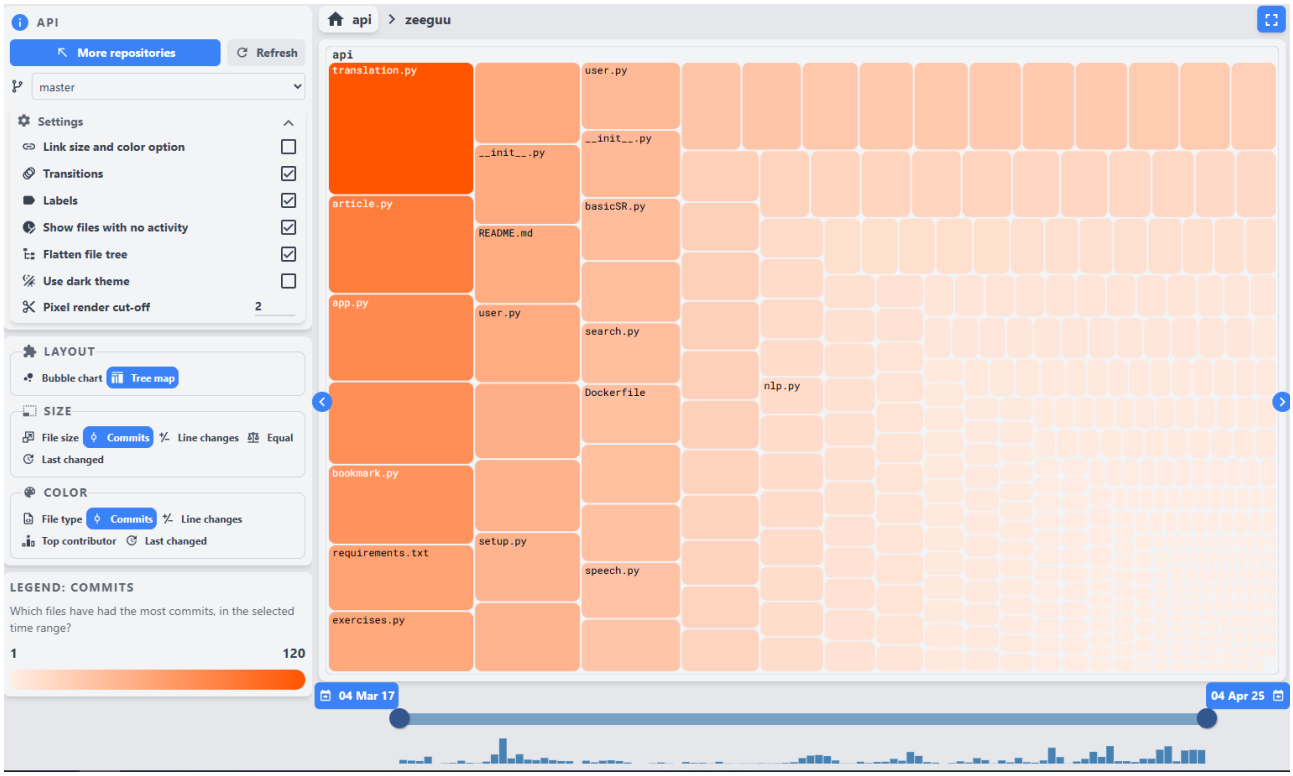


Figure 49: Files receiving the most commits, for all time

Time since file was changed

Figure 50: Time since file was changed for API Zeeguu Project

Top contributors & Code-ownership:

Figure 51: Top contributors for 6 months, for API Zeeguu

Figure 52: Top contributors for last two years, for API Zeeguu

Figure 53: Top contributors for all time, for API Zeeguu

Truck-death number / Files with only 1 author.

Figure 54: Authors who have committed above 95 % of the commits for a file

13.5 Less successful tools

13.5.1 Snakefood:

https://github.com/blais/snakefood

Did not work because: "Python-3.x is NOT supported, but there have been a few efforts to port to it. I never had time to fully port it; it would require a fair bit of a rewrite to do so IMO."

13.5.2 PyCallGraph:

https://pycallgraph.readthedocs.io/en/master/

Couldn't get it to work. Chased down a never-ending rabbithole.

1. Can't find entrypoint / main. Most likely due to the complexity of how the project is run.
2. If I try to help it and give some modules a main(), it then can't figure out the dependencies
3. If I add the dependencies, they haven't been setup properly.
4. Setting them up properly is too hard.

13.5.3 Pyreverse

Had to do cleanup of type annotations in files to make it work.

Gives me a simple package diagram and a class diagram.

Both are huge and unfiltered, and can look a bit like the “spaghetti monster”

13.5.4 Architectural-Lens

https://github.com/Perlten/Architectural-Lens/tree/master

Couldn't get it to generate a non-empty .json file.

- Tried directly through command-line
- Tried through python notebook
- Tried feeding documentation to LLM
- Tried on friends PC

13.5.5 Prospector

https://prospector.landscape.io/en/master/

Gave me some "TreeAttribute" error. Need more time to play around to fix this error

13.6 Optional: (Can SAFELY BE SKIPPED) Recommendations for reengineering of the system (did you discover things that seem wrong and should be corrected?)

I Often see 2 kind of package groupings.
- **Grouped in features** (such as super small micro services, modular monolith, or vertical slices). In short we try to make packages where the modules inside the packages mostly only depend on each other. Often each endpoint will be a single vertical slice. A package may then contain multiple subpackages (vertical slices), that can be related such as Users/CreateUser and Users/DeleteUser.
- **Grouped in code type**, such as controllers, infrastructure, etc.. (often n-layer slices).

Right now we often have a max depth of 2: being something like:
Core → folder → files.

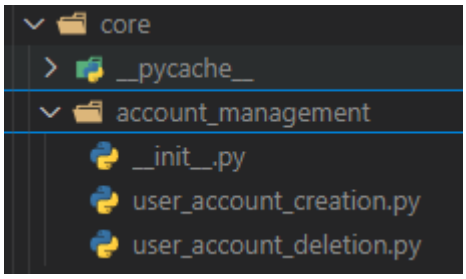


Figure 55: example of core -> folder -> files

As the project grows, it might be beneficial to have more groupings.

Core → Folder for everything content related → Specific content type folder → files

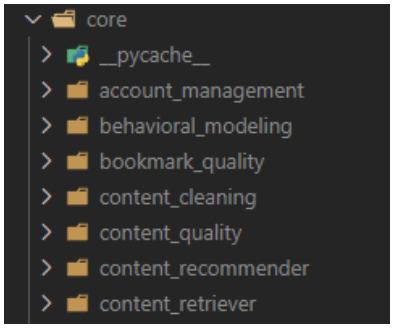


Figure 56: Example of core folder having a lot of different "non-grouped" subfolders

By doing that, we wouldn't have the account_management folder right next to the content_quality folder – but instead have all the Content_X in a package called Content. This might give a better overview of where to work on specific feature.

Architectural Reconstruction of Zeeguu Api

Thor Liam Møller Clausen (tcla)

Introduction

What is the system

I chose to analyse Zeeguu's api repository[1]. It is the backend for their website[2] and is written in python flask. It is a website where users can learn languages similar to Duolingo. More information can be found in the repository's *README* file and on the website.

What is the problem

The Zeeguu api application has over a hundred files. This can be overwhelming for people when they are introduced to the system. The goal of this project is to use abstractions and visualizations to get a better understanding of the general structure of the program, where the different logics are implemented and which parts of the system that are more important in terms of dependency.

Methodology

I want to analyse the python files in the codebase for architectural reconstruction. I want to get an overview of which modules depend on each other, and which are more "central" than others. By that I mean, it has more modules depending on it. This should be extracted as a module view.

Tools

I have chosen to find and use already implemented tools for analysing files and python code. The tools I ended up using are GitTruck[3] and Pyan3[4].

GitTruck

GitTruck gives an overview of files and folders, as well as commit history, file size etc. It works by running the following command which will open a window:

```
# from repository's root folder
npx -y git-truck
```

From here there are several settings to choose from. The goal was to get an initial overview of the file structure of the Zeeguu application. The settings I used are in appendix *A GitTruck settings*. Furthermore, I used GitTruck for file specific statistics like file sizes, commits per file etc.

[1] https://github.com/zeeguu/api
[2] https://zeeguu.org/
[3] https://github.com/git-truck/git-truck
[4] https://github.com/Technologicat/pyan

Pyan3

Pyan3 is a static call graph generator made specifically for python. It can be installed with the following command:

```
pip install pyan3==1.1.1
```

(The newest version had a bug that made it unusable. Hence why I use version 1.1.1)

It uses graphviz[5] which I had to install from their website for it to work.

To generate the graph, I ran the following command from the *Zeeguu* folder:

```
pyan3 *.py */*.py */*/*.py */*/*/*.py -c -G -g -d --dot-rankdir LR --svg > ../../../output/graph.svg
```

The tool does not support recursive wildcard search[6], so I used python paths like **.py* and **/*.py* in the command with enough subfolders to cover all files. The arguments are explained in appendix *B Pyan3 arguments explained*.

The goal was to create a graph that made it easy to read which files and methods/functions call each other. I tried all the different arguments the tool had and ended with these because it best fulfilled the goal.

[5] https://graphviz.org/

[6] The path ***/*.py* can sometimes be used to recursively search subfolders for python files.

Results

GitTruck overview

GitTruck was very useful for getting an overview of how files were arranged.

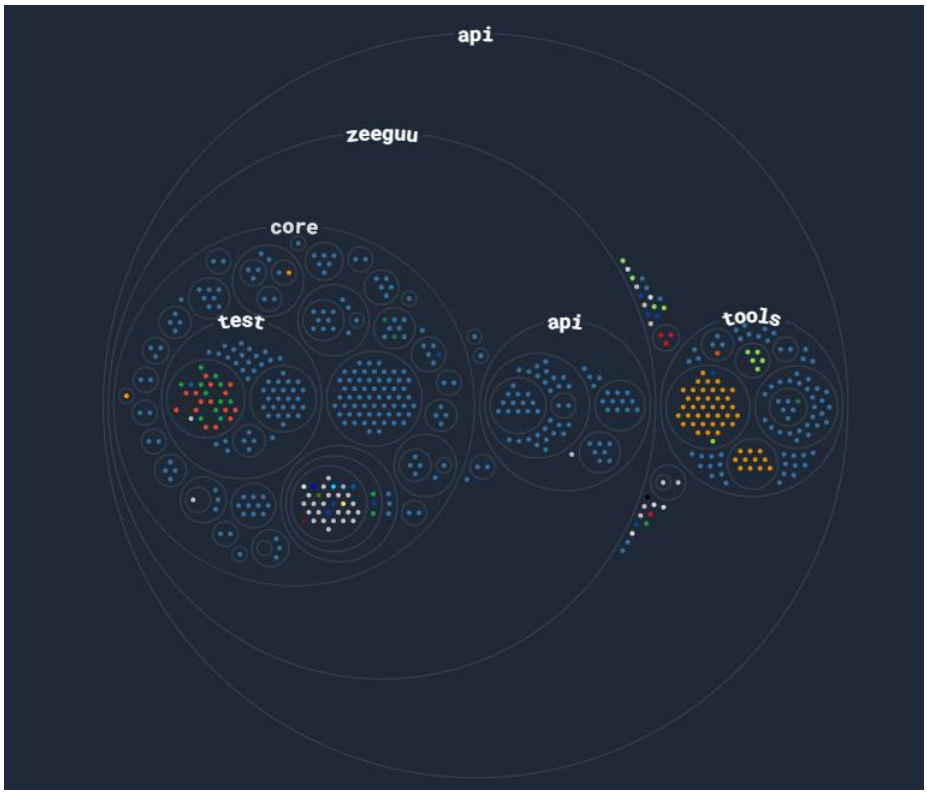


Figure 1 GitTruck bubble chart of Zeeguu's api repository.

After browsing the chart in Figure 1, I made an abstraction with the most important parts of the file structure for my own overview. It can be found in the appendix *C Boiled down file structure of Zeeguu*.

Pyan3

The initial graph it generated was huge and messy with overlapping nodes, which meant some filtering and abstraction was necessary. Based on the overview I got from GitTruck I decided not to include the *tools* folder and focused on the *Zeeguu* folder since it seemed most of the application logic was located there, while the *tools* folder was more about database connections, shell scripts, report generators etc. I also abstracted the test folders away.

Areas of interest

The graph was still too big to fit on a screen, so I browsed it and identified some areas of interest based on incoming calls.

Pyan3 call graph legend
Grey box: File/class containing functions/methods
Coloured node: Function/method/class
White node: Folder/file
Solid line arrows: Arrow from A to B means the method/function A calls method/function B.
Dotted line arrows: Arrow from A to B means A defines B. (Class/function/method with the ***def*** keyword has an outgoing defines arrow)

A module in *core* with a lot of incoming call dependencies (Figure 2):

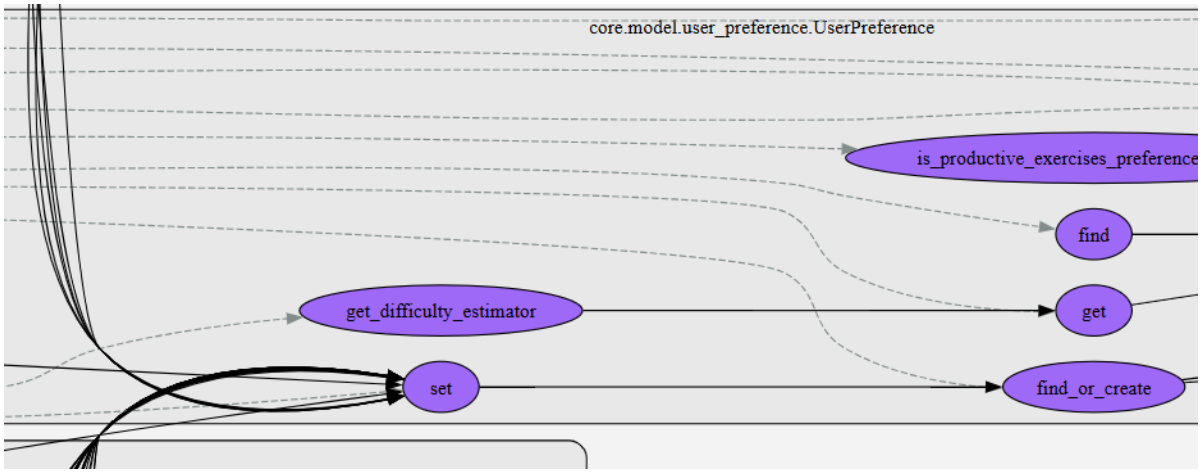


Figure 2 UserPreference "set" method from pyan3 call graph.

A module in *api* with a lot of incoming call dependencies (Figure 3):

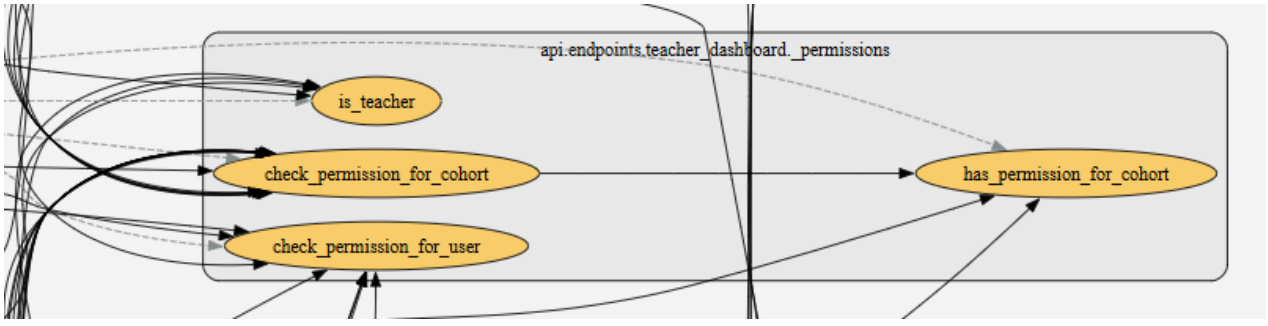


Figure 3 Functions in the permissions file from pyan3 call graph.

Module view based on Pyan3 call graph

For further abstraction I decided to make my own smaller graphs with the important highlights. I took the most used functions/methods based on incoming calls and traced the callers to extract the dependencies. They can be seen in Figure 4 and Figure 5.

Module view legend
Box (module): File with file name and file size in kilobytes.
Arrow (dependency): Arrow from A to B means A has a call to B.
Arrow thickness shows the number of functions from A calling functions from B also denoted by the edge weight (no weight means weight=1).

Core files

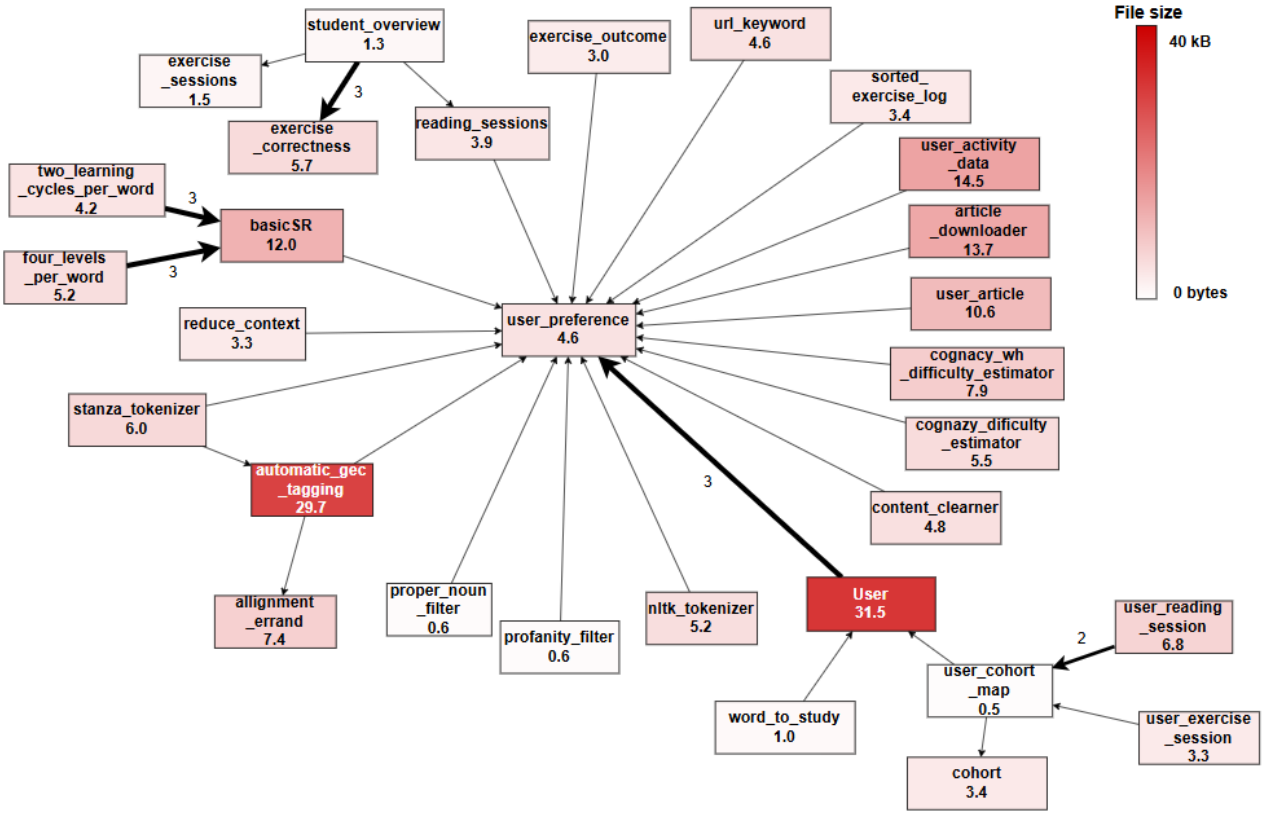


Figure 4 Module view for the files located in the core folder. File sizes were read from GitTruck.

Api files

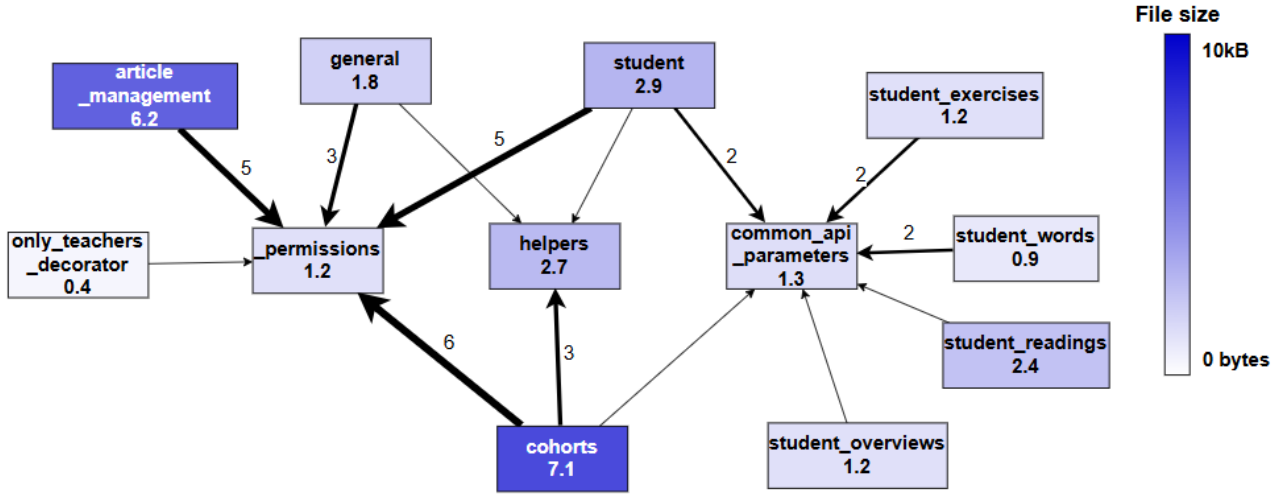


Figure 5 Module view for the files located in the api folder. File sizes were read from GitTruck.

Pyan3 conclusion

After browsing the pyan3 graph it became clear that the app has a lot of logic centralized around users in the core. The api mostly showed calls to helper functions and *_permissions*. The graph does not say much about the files other than the static function/method calls, so I decided to analyse some of them further on GitTruck.

GitTruck file analysis

In GitTruck I hid some of the non-python files, and it turns out the *user.py* file in the *core* folder is the largest python file in the repository, and it was last edited just 26 days ago (from April 23. 2025), which if you order files by last changed puts it in 15th place out of over a hundred files. This can be seen in Figure 6.

Figure 6 Zeeguu api files ordered by file size and coloured by last changed using GitTruck.

Furthermore, the *user.py* file has 52 commits, which puts it in 10th place of files with most commits as seen in Figure 7. The *user_preference.py* file was further down on all GitTruck metrics and was for instance edited 96 days ago.

Figure 7 Zeeguu api files ordered by commits and coloured by line changes using GitTruck.

The file with most commits is *language.py* in the *endpoints* folder which is also in top three with the other metrics except file size. However, the pyan3 graph does not show any dependencies for the file as seen in Figure 8.

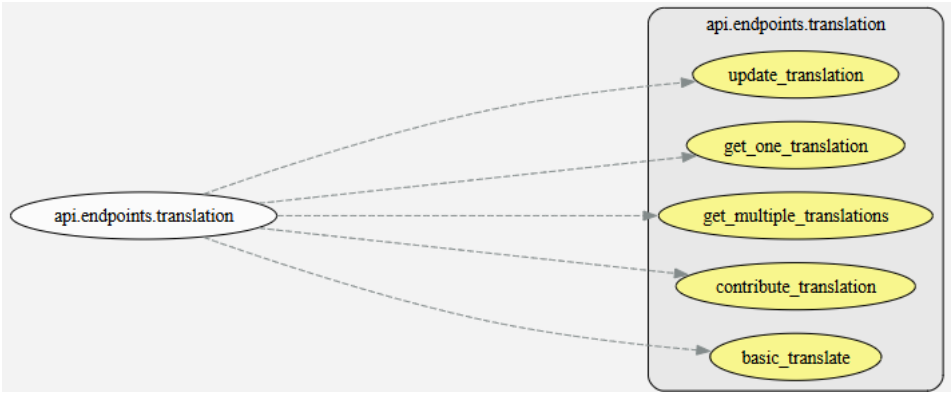


Figure 8 The translation.py file from pyan3 call graph.

Discussion

Tool flaws and limitations

The pyan3 tool had some major flaws that affects the usefulness of the results.

Fails to detect some method/function calls

```
@api.route("/user_preferences", methods=["GET"])
@cross_domain
@requires_session
def user_preferences():
    preferences = {}
    user = User.find_by_id(flask.g.user_id)
    for each in UserPreference.all_for_user(user):
        preferences[each.key] = each.value
```

Figure 9 Code section from user_preferences file in endpoints folder.

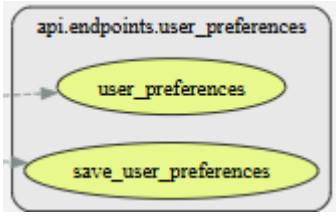


Figure 10 The user_preference file in the pyan3 call graph.

As seen in Figure 9 the ***user_preferences*** function calls the ***all_for_user*** function. But as seen in Figure 10 the pyan3 graph has no arrow from the ***user_preference*** node to ***all_for_user***. In general, there are no edges between files in the *api* folder and the *core* folder. This has led to a graph with a lot of disconnected subgraphs, which forced me to focus on a small part of the core and the api separately, and even here it is uncertain if dependencies are missing.

Limited customization and filtering

The pyan3 tool takes paths as arguments which it analyses, but it does not support exclusion of folders or files. For example, I had to delete the *tests* folder to avoid including it. It was also not possible to edit the graph with the tool, like removing nodes without in or outgoing edges or rearranging nodes to better fit them in a frame.

Improvements if I had more time

The results helped me get an overall idea of what the Zeeguu app contains in terms of functionality. However, pyan3's major flaws made it hard to tell how much important information was missing. The lack of freedom for graph customization and filtering also lead to a lot of manual work. For future analysis I would spend more time finding better tools with more filtering and customization options to avoid a lot of the manual work that went into boiling down a huge graph.

Appendix

A GitTruck settings

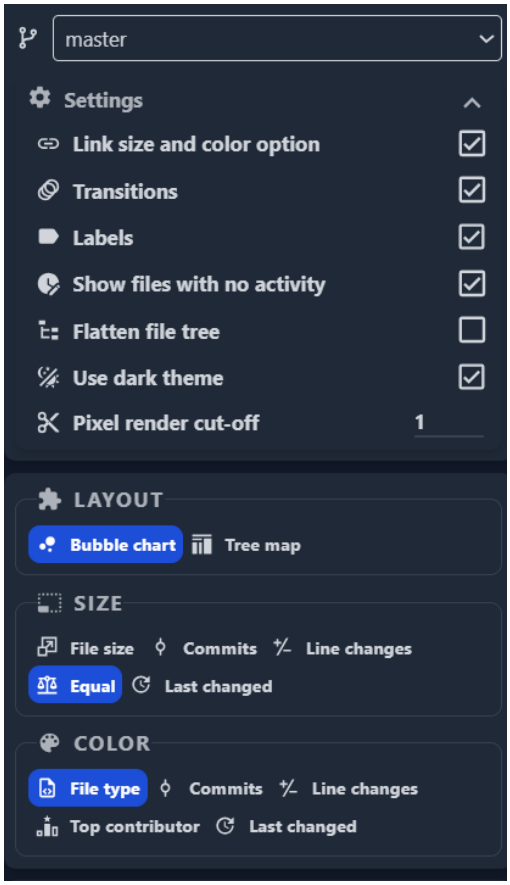


Figure 11 GitTruck settings used for browsing the folder/file structure of Zeeguu's api.

B Pyan3 arguments explained

Taken from the output of running ***pyan3 -h***:

-c, --colored *color nodes according to namespace*

-G, --grouped-alt *suggest grouping by adding invisible defines edges*

-g, --grouped *group nodes (create subgraphs) according to namespace*

--dot-rankdir RANKDIR *specifies the dot graph 'rankdir' property for controlling the direction of the graph. Allowed values: ['TB', 'LR', 'BT', 'RL']*

C Boiled down file structure of Zeeguu

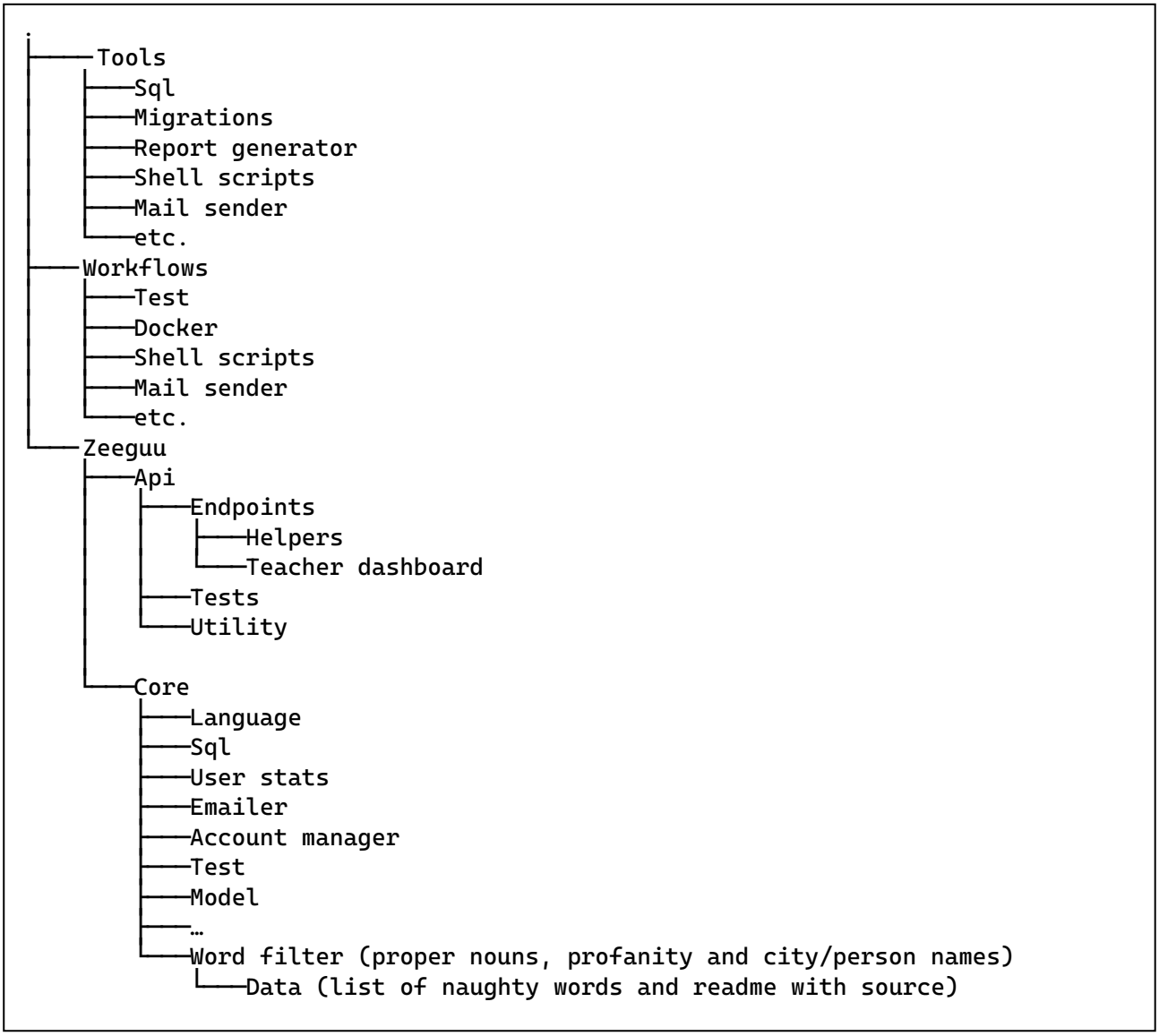


Figure 12 Abstracted file/folder structure for the Zeeguu api repository.

D Time allocation

I spent a significant amount of time trying out different tools that I could not get to work, or did not give me a result that I could use (for example *prospector*[7] that turned out to be more like a linter).

pycallgraph[8] and *pyreverse*[9] gave the same error:
AttributeError: 'TreeRebuilder' object has no attribute 'visit_typealias'.

After approximately two working days I had two tools that worked, pyan3 and GitTruck. Then I spent most of a working day on playing around with these until I felt I had a good overview of the Zeeguu

[7] https://prospector.landscape.io/en/master/
[8] https://pycallgraph.readthedocs.io/en/master/
[9] https://pylint.readthedocs.io/en/latest/additional_tools/pyreverse/index.html

system. I spent another two working days putting my mind on paper - Creating abstractions for file structure and the module view as well as writing down notes and collecting it in a document for the report.

IT UNIVERSITY OF COPENHAGEN

Deliverable 5: (Individual) Architectural Recovery

Master of Science in Computer Science
IT-University of Copenhagen

Course Name: Software Architecture
Course Code: KSSOARC2KU
Submission Date: 16th May 2025

Author	**Email**
Rasmus Ole Routh Herskind	rher@itu.dk

Table of contents

1 Introduction

This report aims at doing an Architectural Recovery for the Zeeguu API Backend: *Zeeguu-API is an open API that allows tracking and modeling the progress of a learner in a foreign language, with the goal of recommending paths to accelerate vocabulary acquisition.*[1]
The report addresses the lack of architectural documentation in the system by reconstructing its structure using a module view, and conducting static analysis to identify key components, their relationships, and responsibilities.

2 Methodology

2.1 Tool support

To do the recovery, I have mainly used the Python scripts provided in the course, along with additional ones to provide different views of the architecture. Some of them I have modified a bit, but their goal stays the same: get hierarchical views of the modules in the source code and integrate new evaluation metrics as described below.

2.2 Data gathering and knowledge inference

The scripts all focus on statically analyzing the API modules by drawing dependency graphs between them. Each node represents a given module, with directed edges illustrating a dependency from one module to the other, which are found using simple regular expressions from the root content folder. To facilitate the creation of the graphs, the following metrics were used:

- Lines of Code (LoC). The bigger the node size, the higher the amount of LoC.
- Code Churn (lines of code modified across all git commits, based on the specific module view). This is illustrated by a viridis color map, going from purple (low churn) to yellow (high churn).
- Module hierarchy, by viewing module dependencies at different levels.

Using the general scripts provided in class and the above-mentioned metrics, I have retrieved the source view presented in figure 1[2].

As the view is rather difficult to get any meaningful information out of at first glance, I will, throughout the results section, section 3, dive deeper into module-based views to get a better understanding of the Zeeguu-API. This was an iterative process, hence only the last iterations are shown in the source code and this report.

[1] Mircea, 2025.
[2] For a bigger view of the image, view figure 6

In general, I have extended the provided scripts with some helper functions for creating new graphs. Some of these functions include the functionality to abstract a provided graph to a given depth, filter the graph (e.g., removing specific nodes), or zoom into a specific node. From the source view presented in figure 1, I will present the top-level view of the system before iteratively exploring different modules while still preserving the graph-based module view.

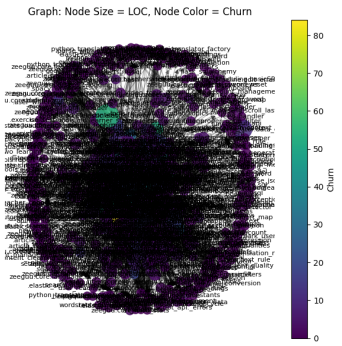
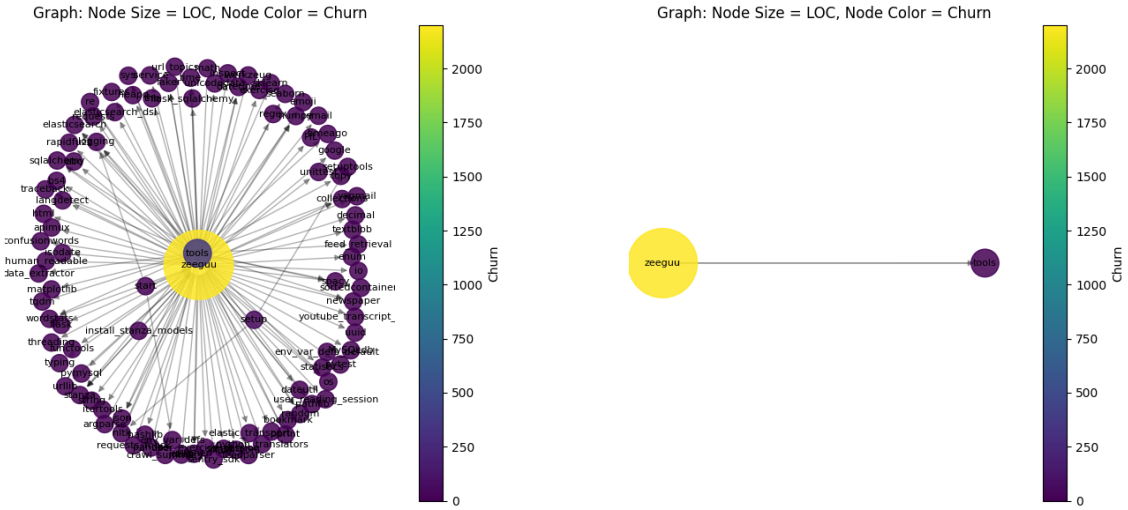


Figure 1: Source view of Zeeguu-API

3 Results

To get a more structured view on the system, I started by getting the top view of the system. However, it was quite clear that the system had a lot of external dependencies, which even further clutter the views. By filtering those out, as illustrated in figure 2b, we see that the system at the root level has two core components: *zeeguu* and *tools*:



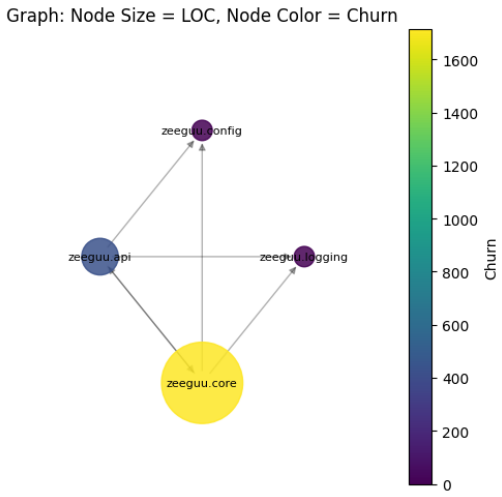
(a) Top view with external dependencies

(b) Top view with only internal dependencies

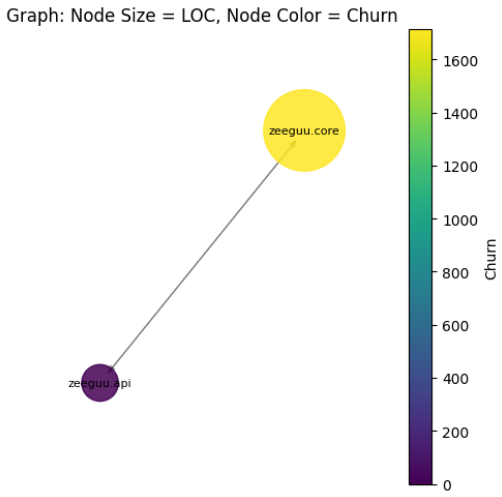
Figure 2: Top view of Zeeguu API

Given the *zeeguu* module scored high in terms of Churn and LoC, I looked further into it, as shown in figure 3, by expanding the depth level to 2. The module mainly consists of *zeeguu.core* and *zeeguu.api*. There are also some dependencies towards *zeeguu.config* and *zeeguu.logging*, but since they both have a low LoC and Churn, I filtered all the next graphs to only show relevant modules (which I defined to be modules with a specific LoC and Churn[3]).

[3]The values for each graph were found through trial and error. Other values could lead to different results, which is also discussed in section 4



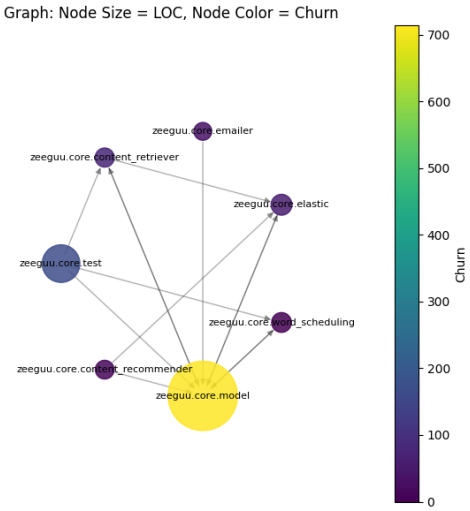
(a) All modules



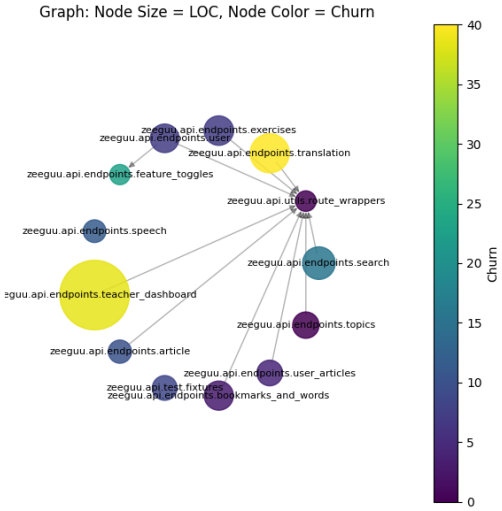
(b) Modules with $Churn > 50$ and $LoC > 50$

Figure 3: Zeeguu module at depth 2

Looking even deeper into the relevant *zeeguu.core* and *zeeguu.api* modules at depth 3 and 4 respectively, we can see how modules with either high churn or a reasonably high amount of lines of code depend on each other in the two modules:



(a) Zeeguu.core module at depth 3 with $Churn > 50$ and $LoC > 50$



(b) Zeeguu.api module at depth 4 with $Churn > 10$ and $LoC > 50$

Figure 4: Zeeguu core and API modules

Interesting to note is how the API modules rarely depend on each other internally, which is in vast

contrast to the core module. Here, a lot of the modules show internal dependencies, indicating a strong coupling between them. Furthermore, the churn in the core module—especially the *zeeguu.core.model* and *zeeguu.core.test* modules—is much higher than the highest churn in the API module. This clearly indicates that an even more fine-grained look at these two core modules could be relevant.

Lastly, I present a view showing how the API and core modules relate to one another. Here, we see that most underlying modules in the API and core depend on each other, which clearly opens a path for future work considerations.

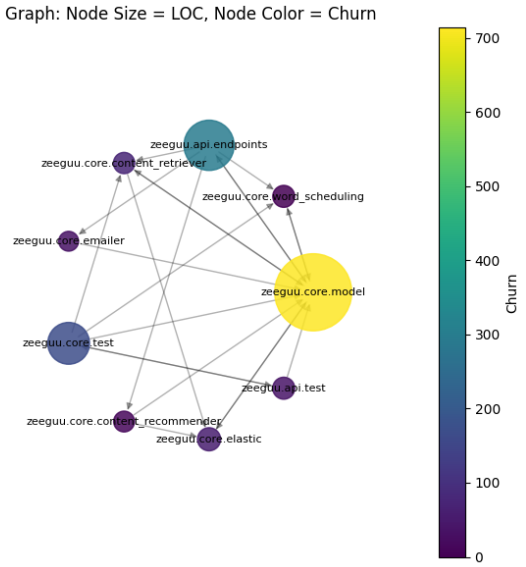


Figure 5: Zeeguu module at depth 3 with $Churn > 50$ and $LoC > 50$

4 Discussion and Conclusion

The main limitation of this project has also been its strength: the metrics used. I have, throughout the script-writing, filtered out specific modules solely based on lines of code and churn of the specific files. Some files and/or even modules could be relevant despite not scoring high in these metrics. I did try many different values of both churn and LoC, but the ones I have chosen for this report, I generally found to be the best. It could be interesting to also have included more metrics, for example a minimal amount of imports.

Given more time, I would have liked to create some more visually appealing diagrams showcasing the views, e.g., UML diagrams. This would help newcomers to the Zeeguu project understand the project and its components more easily, instead of being shown relatively abstract graphs.
It would also be interesting to dive even further into more modules, e.g., the *zeeguu.core.model* module. The complexity, both from LoC and churn, is quite high, which probably also means

rather complex code and potentially room for improvement. This is generally true for the entire project, which is also a clear limitation of my method, given I only went as deep as level 3/4.

4.1 Conclusion

In the report, I have shown a high-level view of the Zeeguu-API backend through inspecting the module hierarchy, lines of code, and code churn. The project mainly consists of two modules from the content root: zeeguu and tools. Given that zeeguu had both a higher number of lines of code and churn, I inspected it even further. This showed rather tight couplings between its API and core module, which would be interesting to look even further into.

A Link to code

All code used during the iterative recovery progress has been written using Google Colab and can be viewed in this notebook.

B Time Allocation

Most of my time was spent writing, testing and iterating thought the scripts on Google Colab. Throughout the process I wrote the report with new discoveries. In total I have spent aroung 60-70% of my time writing scripts, 5-10% on anylysing the results and 20-30% writing the report.

C Figures

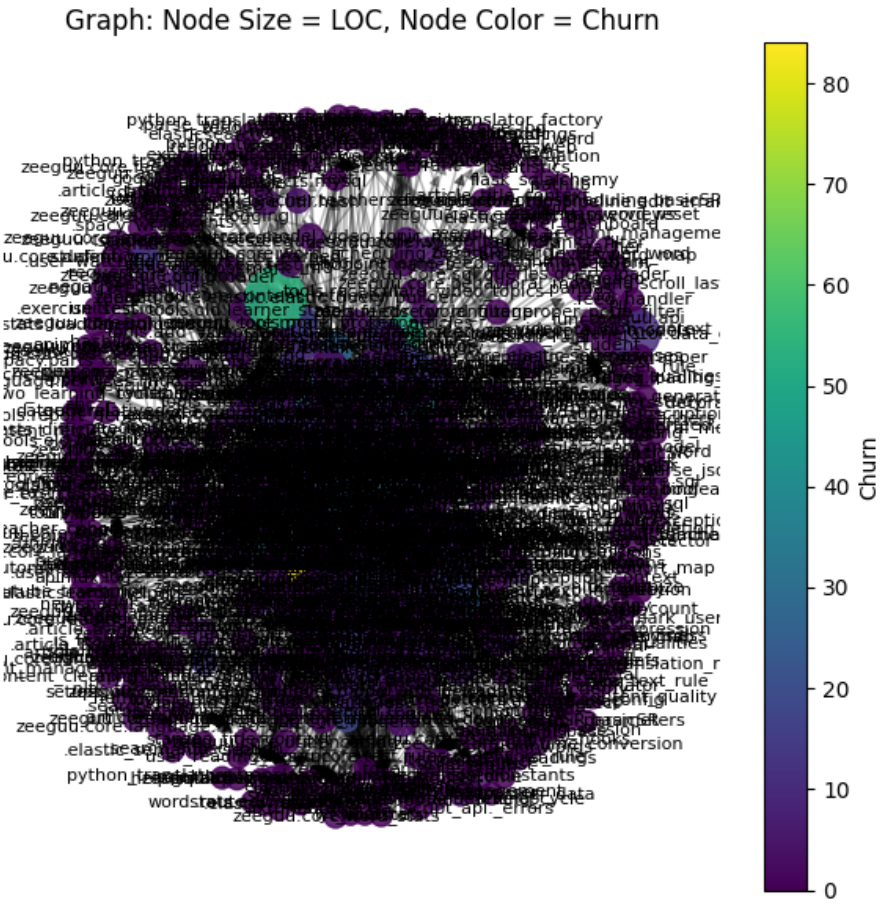


Figure 6: Source view of Zeeguu-API

References

Mircea, L. (2025). Zeeguu api [Accessed: 2025-05-11].