## The Brief

ING is creating a persistent chat application aimed at developers, similar to Slack, called Linus. Users will be able to pick a chat room, in which they can send and receive messages, to and from other people in the chat room.

Several teams will be working on different aspects of Linus which means you are not required to implement every aspect of the application. For example, authentication and authorization is being developed by another team, to be integrated at a later date.

## The Vision

### Scenario 1 – View messages

As a Linus user, I would like to choose a chat room so my conversation is specific to my interests. For example, ‘web development’ or ‘release management’.

As a Linus user, when I join the chat room, I want to see a history of the messages previously posted to the chat room. Consider these rooms maybe long-lived and contain hundreds or thousands of messages.

As a Linus user, I would like to see messages sent from other users in my chat room, in near-real-time.

As a Linus user, I want the messages in my chat room to be labelled with the username of the person who posted the message and the datetime it was posted, so people know who said what and when.

As I Linus user, I would like to send emojis.

### Scenario 2 – Publish a message

As a Linus user, I want to post messages to my chat room so that other users can see my message.

As a Linus user, I do not want my messages modified after they have been published to the chatroom.

### Scenario 3 - Search

As a Lionel user, I would like to search chatroom messages for specific text or phrases.

### Scenario 4 - Redaction

As the fun-police, I would like profanities to be replaced with \*something\* before they are posted to the chat window. It may help to know that Google provides a service to identify profanities: <http://www.wdylike.appspot.com>

## Expectations

There are many things you should consider, even if you don’t have time to implement them all.

You should consider this when developing the UI:

* Accessibility
* Will you support multiple form factors?
* What about the offline experience?
* Have you considered how you will manage state/data?

API development should consider:

* How do you deal with large data sets?
* Do you have adequate logging?
* If you’re using an external profanity API, what happens if the external service is down?
* How do you design your API to be RESTful
* Is this application suitable for TDD or BDD or DDD or some other \*DD?

Testers, think about:

* Bang for your buck; which tests give you the highest value for the effort you put into them
* What about non-functional testing, like load or stress testing?
* How will you populate the application with test data? This may help: <https://github.com/toddmotto/public-apis#test-data>
* Please use BDD-style testing, Cucumber is your friend
* Create a test report:
  + Simple & Customised HTML Test Report with the following attributes
    - Feature Name, No of Test passed, No of failed, Total pass %
    - How would you manage screenshot on failure?
* Try using cucumber Extend report and link screenshot on failure in each step.

Scrum masters:

* It’s better to demonstrate one thing working well, than it is to demonstrate everything half-complete
* Use your product owners
* Be open and honest with your stakeholders and yourselves; if you’re not going to make it, change direction or negotiate scope with your product owners
* Protect your team from themselves in case they are overambitious and overcommit

## Review

Don’t underestimate the importance of preparing for the review. Each team will have 15 minutes to demonstrate what they have done. Consider finishing development 15 minutes before reviews are scheduled so you have time to do a practice run.