NB: All in-paper notes will be marked with the word TODO in upper case letters.  
TODO: Frontpage goes here! Fix in a pdf creator?

# Abstract

Contents

[Abstract i](#_Toc360290225)

[Preface iii](#_Toc360290226)

[1 Introduction 1](#_Toc360290227)

[1.1 Background 1](#_Toc360290228)

[1.2 Essaychapters.. 1](#_Toc360290229)

[1.3 Problem statement 1](#_Toc360290230)

[1.4 Outline 1](#_Toc360290231)

[2 Methodology 1](#_Toc360290232)

[2.1 Selection criteria 1](#_Toc360290233)

[3 Part 1 (change this) 1](#_Toc360290234)

[4 Part 2 (and this) 1](#_Toc360290235)

[5 Results? Or under each part? 1](#_Toc360290236)

[6 Conclusion 1](#_Toc360290237)

# Preface

# Introduction

TODO: most of essay will be under here.

## Background

From Essay: Background and introduction chapter (minus the “in this paper “stuff)

## Essaychapters..

## Problem statement

## Outline

## Terminology

TODO: Define framework == library in some cases (socket.io, SignalR)  
TODO: Transports == WebSockets, SSE, Long-Polling….  
TODO: WebSockets is not plural, websockets are.  
TODO: IntelliSense

# Methodology

TODO: Where to write how many frameworks??  
TODO: What tense to write in??  
TODO: Have a list over used technologies and reference that instead of footnotes everywhere?

To be able to compare the five frameworks, I need to have a complete impression of each–both what they deliver in form of usability and how they perform. Consequently, the work on(TODO: with?) this thesis will be split into two parts.

In the first part the frameworks will be reviewed from a programmers perspective. This includes aspects like documentation, API, learning curve and other elements concerning general usability.

The second part will look at how well each framework performs. Each framework will be put through a series of load tests for different scenarios. In the end, this will give objective results as opposed to the more opinion-based testing I will do in the first part.

## Selection criteria

Screening for the five frameworks that will be featured in the thesis, will be done by the criteria described in this section. Each of the selected framework will have to stand out from the rest in order to be considered for further research.

TODO: subsections describing each criteria or just a bullet list?

### WebSockets support

A framework does not need to offer WebSockets support in order to be considered, but it must at least mention plans for it, either in a roadmap or somewhere else on the frameworks homepage. However, if WebSockets is not supported, the framework has to offer some unique design or functionality that makes it worthwhile for a deeper study.

### Fallbacks

Supporting as many systems as possible is almost always the goal for computer software. For a web application framework that generally means supporting all major browsers. Certain transports are unavailable to older browsers. Therefore, a good real time framework has fallbacks in order to support as many browsers as possible. If a framework has support for one transport only, it is not eligiblefor this thesis.

### Documentation

Learning something new without reading about it first is generally a near impossible task. Any framework without documentation will not be considered at all–no matter the impression it gives regarding any of the other criteria in this section. If documentation is present, but incomplete, the framework will need to offer something special to be a part of this thesis.

### Presentation

It should be easy to find information about the framework. Furthermore, the information offered should be relevant and not just superfluous text to make it look more appealing. The general impression the framework gives has to be professional, meaning that the homepage, or GitHub page, should not have a lot of flashing lights and other unappealing elements.

### Testing

Being able to write automated tests is crucial to make any application maintainable. A framework therefore needs to give some indication that you can write testable code with it. Any framework that clearly states that it does not support unit tests, has to offer some unique design or functionality for it to be considered.

### Community

The purpose of any software is to be used by someone. Many real time frameworks are brand new, and thus has very small user bases. This is not necessarily negative, but a very new framework is probably not mature enough to be one of the five frameworks I will study. Older frameworks that still has small communities though, will not be selected.

### Other

The following criteria are considered less important, but still count towards the final screening decision:

* **Sessions:** The ability to store session data is not relevant for a library that is meant for direct integration with existing web application frameworks like for instance the .NET Framework.
* **Tutorials and demos:** Though it is preferable to have tutorials and demos to help with the learning process, it is not required.
* **Collaborators:** If a framework is already in use in production code of well known applications, it is definitely an advantage. However, considering that some frameworks might be quite new, the absence of large collaborators is not considered crucial.

## Evaluation process

The evaluation process will consist of the development of a relatively small application that covers all the common use cases of real time applications: simple messaging and broadcasting of messages–all instant.

### Description of application

For each framework, I will implement an auction house, called “Master Auctions”. The application has the following requirements specification:

* Users must receive real time updates regarding all global events.
  + Global events are defined as all actions except from logging in and registering a new user.
* Users must be able to register an account and log in.
* Users must be able to add and remove items.
  + Users can only remove an item added by themselves.
  + An item does at least have the following properties: name, minimum price, info about who added it and who has the lead bid.
* Users must be able to place bids on all items, including their own.
  + Bids lower than the current bid or bids lower than the minimum price should be disregarded.
* If the framework does not specify a specific template engine or other means of creating views, the application will utilize a common view implemented in Knockout[[1]](#footnote-1).
* MySql will be utilized as database unless it requires substantial workarounds, that may cause the framework to misbehave, to implement it.
* The application will be run locally using either the server bundled with the framework, or a server best applicable for it[[2]](#footnote-2).
* Integration tests will be done by the most applicable way. If there are better ways than using a browser to test (like Selenium[[3]](#footnote-3)), it will be used instead.
* All tests (integration and unit) will use common testing frameworks in the framework language. For Java: JUnit, for C#: NUnit, for JavaScript: Mocha with some assertion framework like should.js. TODO: list of technologies

### Discussion of use cases

The use cases, registering user and logging in, will test simple one to one communication between the server and the client. The remaining use cases: adding and removing items and placing bids, will test broadcast of one clients action to all other connected clients.

Real time applications also have one other use case: client to client communication (via the server). So called “peer to peer” communication is not part of the specification of the test application simply because it is a subset of simple one to one communication from client to server. The only real difference is that the outgoing message from the server would go to a different client than the origin. Technically this is not worth testing in the test application. It will, however, be a part of the performance test cases (TODO: kryssref og skriv om det).

### Common UI

To be able to use a client side view framework like Knockout is getting more and more vital in modern web applications. Such a framework handles a lot of UI updating, and generally makes views more maintainable and easy to write. I feel that is necessary to try to keep the same view as far as it is possible. If the framework under test comes bundled with another view engine however, I will use that instead if it is possible.

### Choice of database engine

MySql, while it is an aging database engine, is one of the oldest, best maintained and used database engines on the marked. It is reliable and simple to use, and it should be universal enough for all frameworks to use. If, however, some framework does not support it out of the box, I will have to consider not to use it. Using another database for a specific framework is allowed if and only if making it work with MySql requires some workarounds that may change the frameworks original behavior. All hacks that require changing the frameworks source code is also out of the question, and will lead to the usage of one database engine officially supported by the framework.

### Choice of server

Running each application locally has both pros and cons. The pros are that I will eliminate potential lag caused by network traffic, and thereby ensuring the experience to be equal to using an external server under optimal conditions. Using a server locally also usually requires little or no configuration, which minimizes the probability of errors due to wrong configuration. On the other hand, running externally would ensure that all available recourses (RAM and CPU) are used solemnly by the server. This does not matter that much to the development of the test application, but it may impact performance tests. I will therefore reconsider server for these (TODO: reconsider).

## Evaluation criteria

When working with each test application, I will do a thorough evaluation of the frameworks from a programmers perspective. Part of this evaluation will be a deeper look into the points from section , but other aspects will also be evaluated. This section will describe how this process will unfold.

### Getting started

While it is not a big part of the developer process, it is still an important factor how easy it is to get up and running with some functioning code. I will emphasis how well the installation process is documented, and whether there are demos or examples or not to help you get started. How steep the learning curve is, will also be discussed under this point.

### Coding environment

Does the framework come bundled with an IDE (TODO: forkortelse)? If not, does other, established IDEs support it? Or are you forced to use a basic text editor? Having a good IDE is very useful, especially when working with new technologies. However, it isn’t much help if the IntelliSense support is non-existent. And even more important is debugging opportunities on both client and server. All of these together makes up the coding environment, and I believe having a solid environment is crucial for getting people to use a framework.

### Code structuring

Being able to write maintainable code without having to go through a lot of extra work to do so, is even more crucial than having a solid coding environment. This criteria will cover how easy and naturally the application code can be separated into small units[[4]](#footnote-4). I will also give an evaluation regarding code intrusiveness: Does the framework force developers to apply certain patterns, or is it more free?

### Serialization

Passing data back and forth between a client and a server is usually not a straightforward process. Generally, the client is implemented in one language, and the server in another. Data must then be exchanged in a format that can be understood by both, and that’s where serialization comes into play. (TODO: figur) Introducing a language that both sides can serialize to and deserialize from, makes data exchange more feasible. As this is a common scenario, I will look at how the frameworks handles this process–if it is handled for you, or if you have to do it yourself manually.

### Simplicity

If any part or practice of the framework seems unnecessarily complicated, I will write about it here. Also, if something I expected to be hard is made easy, it will be taken into account under this criteria.

### Revisited criteria

* **WebSockets support:** If the actual support deviates from the impression the initial screening gave, or if the framework does not support it, I will revisit this criteria.
* **Fallbacks:** If fallback support has to be handled manually, or if it just isn’t what was promised, I will need to revise my initial review.
* **Documentation:** When working with each framework, the documentation will most likely be more actively used. Therefore this criteria will always be revisited. Quality of demos and examples will also be written about under this point.
* **Testing:** Being able to write unit- and integration tests are such important aspects of any application that this also will be revised for each framework.
* **Community:** If I have used, or tried to use the community for help during the developing process, I will revisit this criteria.

## Performance testing

TODO: write about the test cases

# Part 1 (TODO: change this)

# Part 2 (TODO: change this)

# Results? Or under each part?

# Discussion

TODO: Gain of using frameworks?  
TODO: When to use real time?

# Conclusion

1. [www.knockoutjs.com](http://www.knockoutjs.com) [↑](#footnote-ref-1)
2. For instance a framework for .NET is natural to run on the Visual Studio Development Server. [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)
4. A unit can be either a module, class or just a single file, depending on language. [↑](#footnote-ref-4)