Архитектура софтверског система

Тим: Стручњаци

Чланови тима:

- Вељко Цоцојевић veljko.cocojevic@elfak.rs
- Јован Јовановић jovanovicjovan@elfak.rs
- Андрија Грбушић strucnjak@elfak.rs

Назив пројекта: **GalaxyUML**

Садржај

Контекст и циљ софтверског пројекта

Контекст

Пројекат се развија као интерактивна колаборативна десктопп апликација која омогућава ккорисницима да у реалном времену додају објекте на цртачку таблу и комуницирају путем чета. Апликација је намењена за онлајн састанке, предавања или тимски рад, где више корисника истовремено жели да ради на истој табли.

Систем функционише у дистрибуираном окружењу, са клијентима који се повезују на централни сервер (Blackboard) или користе message broker за рил-тајм синхронизацију.

Обезбеђује контролу приступа, тако да организатор састанка одређује ко и кад може да мења садржај табле.

Циљ софтверског пројекта

Омогућити да више корисника симултано и у реалном времену цртају и раде на заједничкој табли. Имплементирати рил-тајм чет између корисника, како би могли да комуницирају током цртања.

Обезбедити перзистенцију промена и undo/redo функционалност, како би се сачувала историја рада и избегао губитак података.

Примена архитектурних образаца (MVVM/MVC, Blackboard, push type Event-based Implicit Invocation) ради модуларности, скалабилности и лаког одржавања.

Омогућити једноставно проширење система (нови објекти за цртање, додатне функционалности).

Архитектурно-специфични захтеви

Функционални захтеви

Сви корисници треба да могу да виде таблу и промене на њој у реалном времену. Организатор састанка има иницијално право измене изгледа табле (цртање) и право давања дозволе осталим корисницима да мењају садржај.

Корисници могу да комуницирају путем заједничког чета.

Више различитих објеката који могу да се цртају (додају на таблу).

Постојање више дисјунктних група корисника где свака има приступ својој табли.

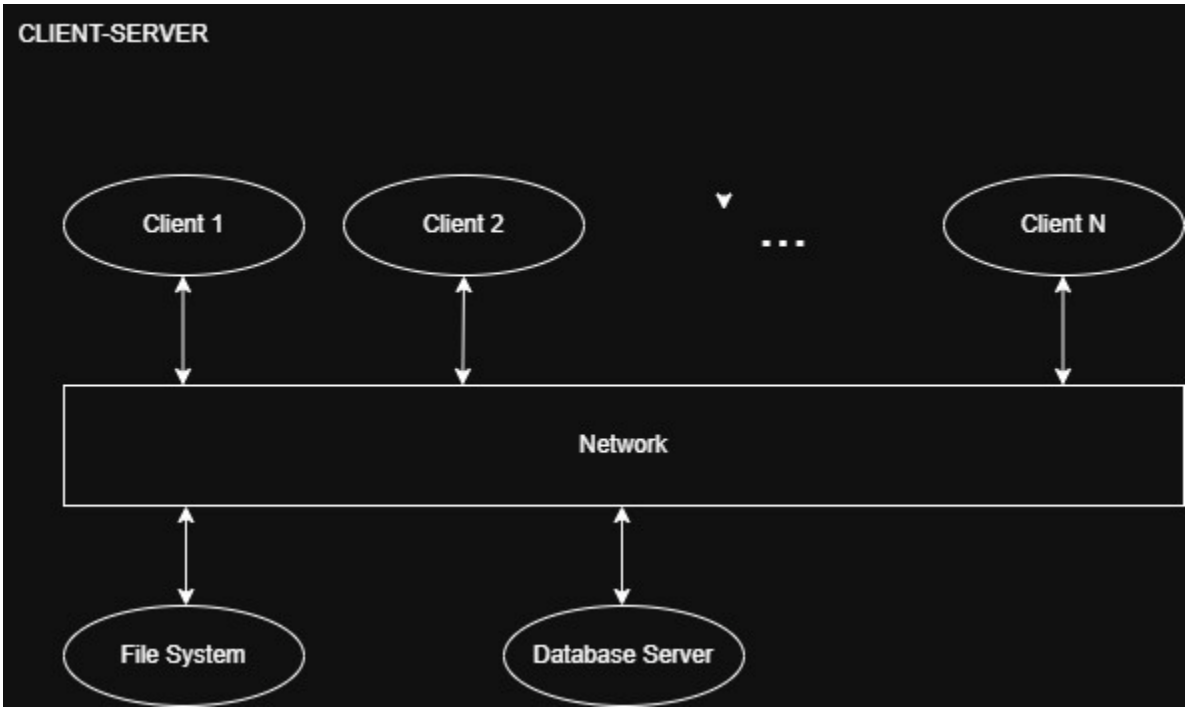
Нефункционални захтеви

Перформансе	Промене које начини један корисник треба да буду пропагиране до свих корисника за највише 2 секунде.
Сигурност	Логовање корисника. Сваки корисник има приступ само табли састанка ком је придружен.
Управљање ресурсима	Ослобађање меморије након брисања објеката са табле. Event processing не сме блокирати GUI. Мрежни проток оптимизован; шаљу се само делта промене по догађају.
Употребљивост	Интуитивни GUI • Канвас и алати за цртање морају бити лако препознатљиви • Дугмад јасно означена • Чет прозор једноставан и прегледан Једноставна интеракција • Повлачење линија, селектовање и брисање објеката интуитивно • Минималан број корака за основне операције Брза реакција система • Промене на табли и нове поруке се приказују брзо • Не блокира се GUI thread Приступачност • Текст читљив, контраст боја добар
Доступност	Теоретски систем може бити увек доступан (докле год сервер ради).
Поузданост	Губљење порука није дозвољено.
Скалабилност	Систем је предвиђен за 10-50 корисника.
Модификабилност	Мора да обезбеди једноставно додавање нових објеката.

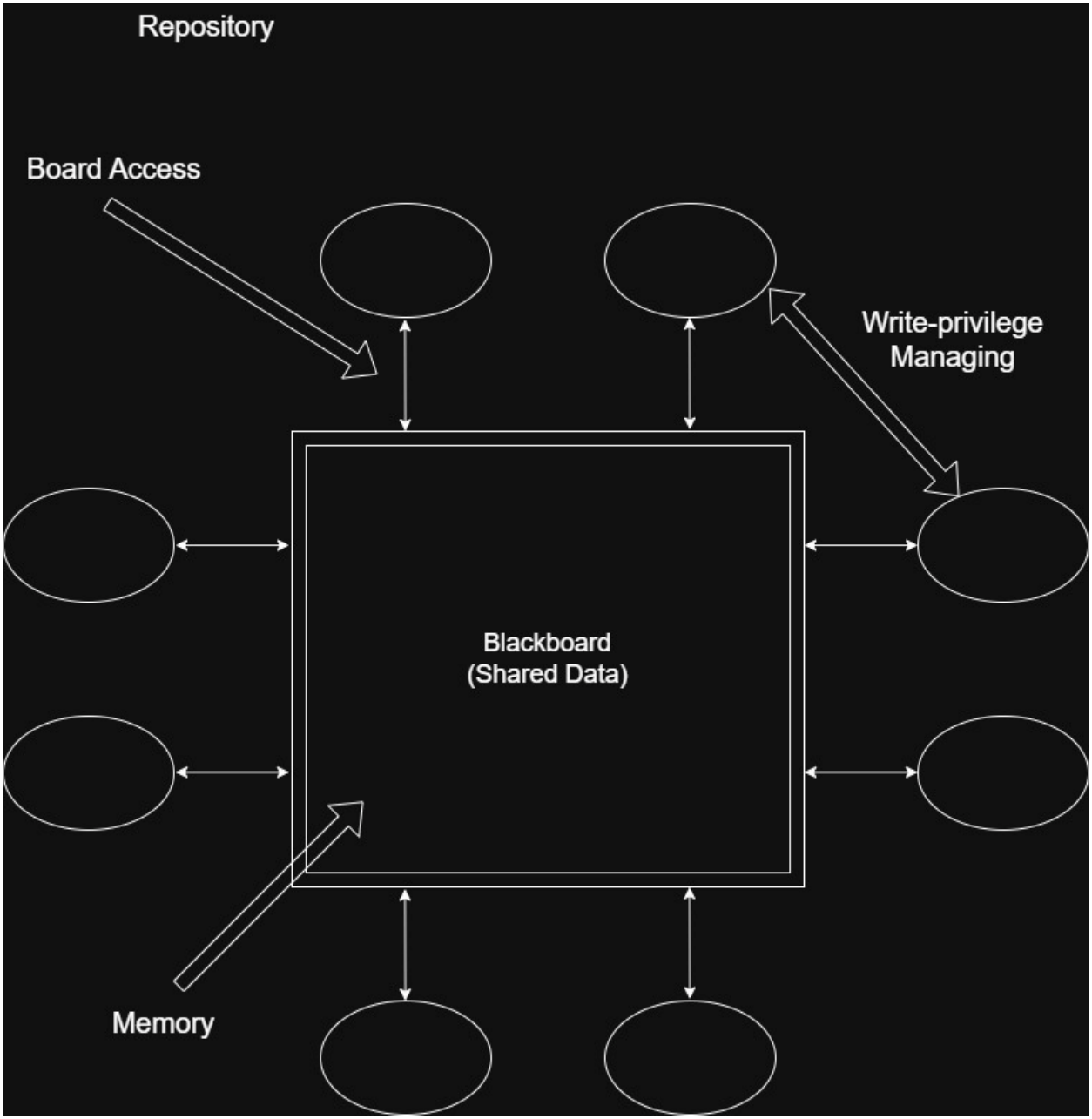
Архитектурни дизајн софтверског система

Архитектурни обрасци

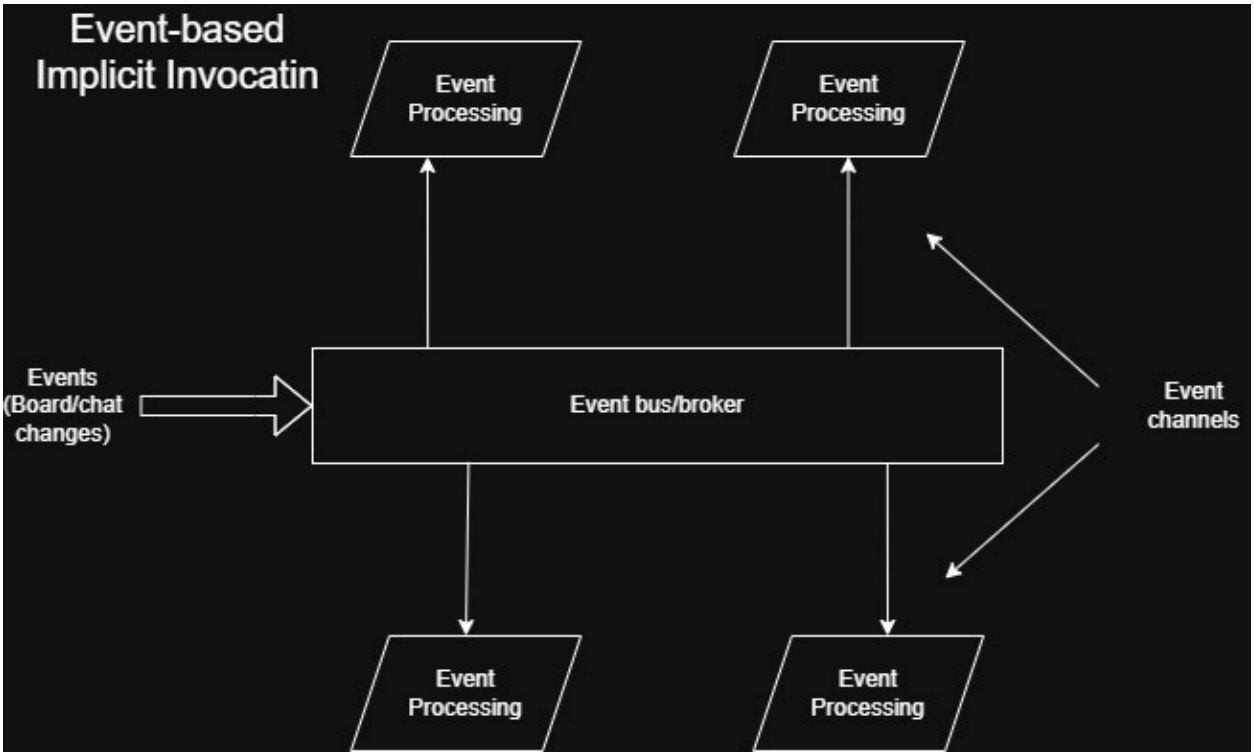
Client-Server – Наша апликација је дистрибуирана апликација.



Repository – Образац за складиштење (дељење приступа). Радимо **Blackboard** (активно складиште).



Event-based Implicit Invocation (push) – Кад клијент начини промену на табли имплицитно се обавештавају сви остали клијенти који приступају истој табли. Радимо **Event-based** подваријанту и то **push** типа јер је циљ да се сви обавесте кад се деси промена и да такорећи цртачка табла одлучује кад ће се то десити (зато је **push**).

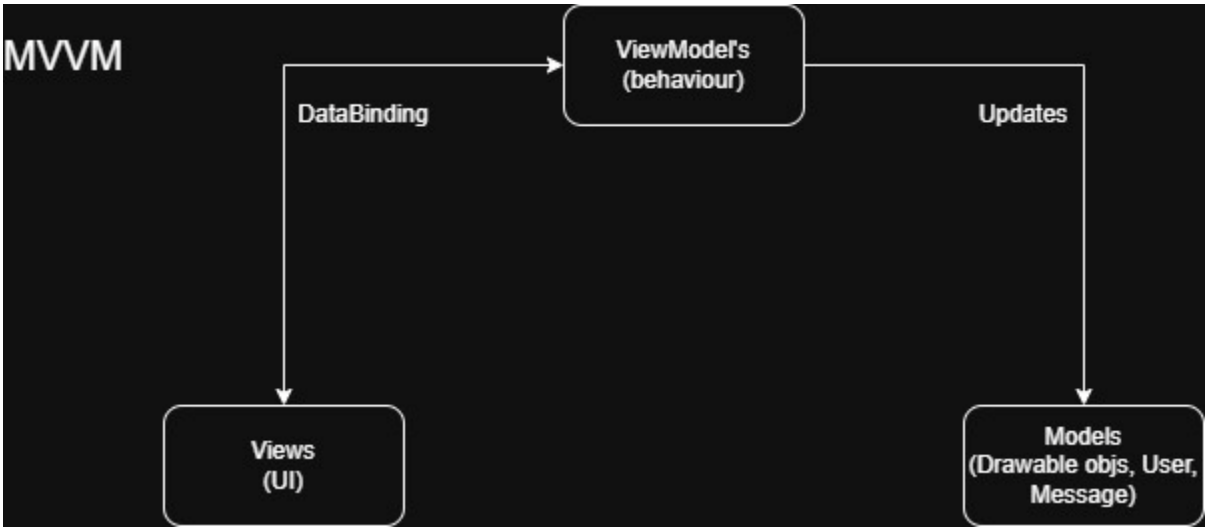


MVVC/MVM

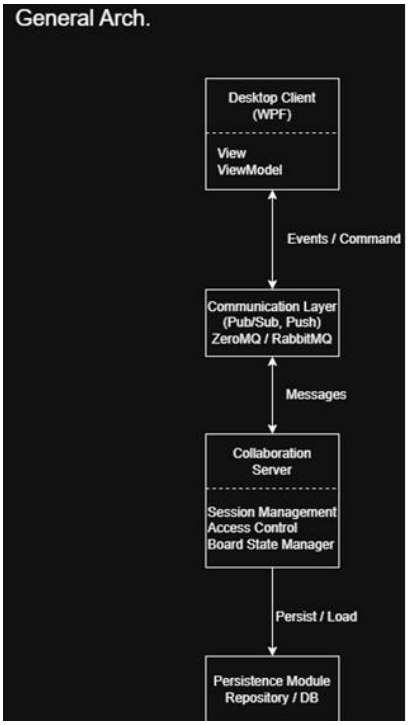
1. Корисник повуче линију на Canvas-у
   (акција мишем у View-у)
2. View региструје догађај и активира Command
   View не садржи пословну логику, већ позива DrawCommand дефинисан у ViewModel-у.
3. ViewModel прими Command
   ViewModel проверава да ли корисник има право цртања и припрема податке за нову линију.
4. ViewModel мења Model
   Креира се нови објекат типа Line, који се додаје у ObservableCollection<Shape>.
5. ViewModel шаље догађај серверу
   Промена се пакује у event поруку и шаље преко Communication Layer-а (publish/subscribe, push варијанта).

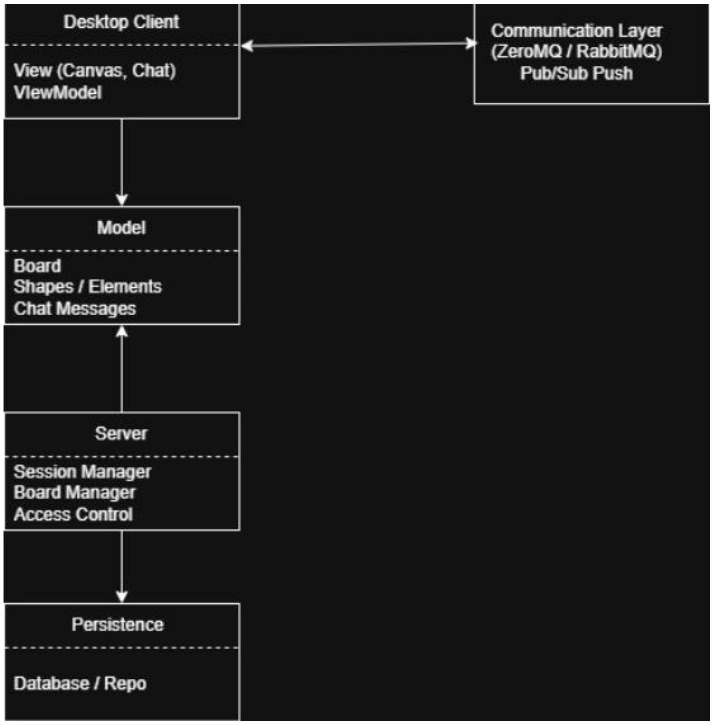
6. Model је промењен → View се аутоматски освежава
   Захваљујући data binding механизму, View аутоматски приказује нову линију без ручног освежавања.

7. Остали клијенти добијају догађај (push)
   Њихов ViewModel ажурира Model, након чега се њихов View аутоматски освежава.



Генерална архитектура система и структурни дијаграм





Техничка и пословна ограничења

Техничка ограничења

Клијентска апликација мора бити имплементирана као desktop апликација, коришћењем C# и WPF технологије.

За организацију корисничког интерфејса и логике мора се применити MVVM архитектурни образац.

Комуникација између клијената и сервера мора бити реализована коришћењем message-passing библиотеке или message broker-а (нпр. ZeroMQ или RabbitMQ).

Серверска компонента мора подржавати event-based publish/subscribe комуникацију у push варијанти.

Систем мора бити компатибилан са Windows оперативним системом.

Перзистенција података мора бити реализована коришћењем централизованог repository-ја / базе података.

GUI нит не сме бити блокирана мрежном комуникацијом или обрадом догађаја (асинхрони рад).

Пословна ограничења

Пословна ограничења дефинишу оквир у ком се систем развија и користи, независно од техничке имплементације.

Пословна ограничења система

Систем је намењен едукативној и демонстрационој употреби у оквиру академског пројекта.

Обим функционалности је ограничен на:

заједничко цртање UML/дијаграмских елемената

текстуални chat

управљање улогама корисника

Максималан број истовремених корисника у једној сесији је ограничен на 10–50 корисника.

Аудио и видео комуникација нису део основне функционалности система (могућа будућа надоградња).

Систем не захтева интеграцију са спољним комерцијалним сервисима.

Безбедносни механизми су основног нивоа (аутентикација и контроле приступа), без напредних enterprise механизама.

Рок за испоруку архитектурне фазе је дефинисан наставним планом и временским оквиром курса.

Изабрани апликациони оквири и библиотеке

1. Клијентска компонента (desktop апликација)

Језик: C#
Framework: WPF (Windows Presentation Foundation)

Разлог избора:

- Модеран графички кориснички интерфејс са подршком за Canvas, toolbar и стилизоване контроле.
- Једноставна имплементација **MVVM обрасца (Model–View–ViewModel)**, што омогућава јасну сепарацију GUI-ја и логике.
- Одлична подршка за **data binding**, што омогућава аутоматско освежавање View-а при променама у Model-у.

Обрасци који се примењују у клијенту:

- **MVVM образац:**
  - **Model:** чува стање табле, chat порука и корисничких права.
  - **ViewModel:** прима догађаје из View-а, ажурира Model и обрађује долазне event-е са сервера.
  - **View:** Canvas за цртање, chat прозор и toolbar, који се аутоматски освежавају при променама у Model-у.
- **Event-based push:**
  Клијент се претплаћује на догађаје са Blackboard сервера или message broker-а и реагује када стигну нови догађаји.

Библиотеке и алати:

- **Reactive Extensions (Rx.NET)** – за елегантно управљање догађајима и асинхроним токовима података.

- **Newtonsoft.Json** – за серијализацију објеката (цртежа и порука) приликом слања преко мреже.
- **ZeroMQ или RabbitMQ .NET клијент** – за комуникацију са сервером / broker-ом.

2. Серверска компонента (дистрибуирани сервер / Blackboard)

Језик: C#
Тип апликације: конзолна или сервисна (.NET)

Обрасци који се примењују:

- **Blackboard / Repository образац:**
  Централизовано активно складиште стања табле, историје цртежа и chat порука.
- **Implicit Invocation / Event push:**
  Сервер (Blackboard) одлучује када ће клијентима послати нове догађаје.

Библиотеке и алати:

- **ZeroMQ / RabbitMQ .NET клијент** – за дистрибуцију догађаја ка свим клијентима.
- **Entity Framework Core** – за перзистенцију података у локалној или удаљеној бази података.

3. Додатне напомене

- **MVVM образац** се примењује искључиво на клијентску GUI логику, док сервер користи **Blackboard + Event push архитектуру**.
- Оваква комбинација омогућава:
  - Јасну модуларност и сепарацију одговорности
  - Једноставно проширење система (додавање нових типова објеката или догађаја)
  - Поуздану и real-time дистрибуцију догађаја свим корисницима

Контекст и циљ софтверског пројекта

Контекст

Пројекат се развија као интерактивна колаборативна десктопп апликација која омогућава ккорисницима да у реалном времену додају објекте на цртачку таблу и комуницирају путем чета. Апликација је намењена за онлајн састанке, предавања или тимски рад, где више корисника истовремено жели да ради на истој табли.

Систем функционише у дистрибуираном окружењу, са клијентима који се повезују на централни сервер (Blackboard) или користе message broker за рил-тајм синхронизацију.

Обезбеђује контролу приступа, тако да организатор састанка одређује ко и кад може да мења садржај табле.

Циљ софтверског пројекта

Омогућити да више корисника симултано и у реалном времену цртају и раде на заједничкој табли. Имплементирати рил-тајм чет између корисника, како би могли да комуницирају током цртања.

Обезбедити перзистенцију промена и undo/redo функционалност, како би се сачувала историја рада и избегао губитак података.

Примена архитектурних образаца (MVVM/MVC, Blackboard, push type Event-based Implicit Invocation) ради модуларности, скалабилности и лаког одржавања.

Омогућити једноставно проширење система (нови објекти за цртање, додатне функционалности).

Архитектурно-специфични захтеви

Функционални захтеви

Сви корисници треба да могу да виде таблу и промене на њој у реалном времену. Организатор састанка има иницијално право измене изгледа табле (цртање) и право давања дозволе осталим корисницима да мењају садржај.

Корисници могу да комуницирају путем заједничког чета.

Више различитих објеката који могу да се цртају (додају на таблу).

Постојање више дисјунктних група корисника где свака има приступ својој табли.

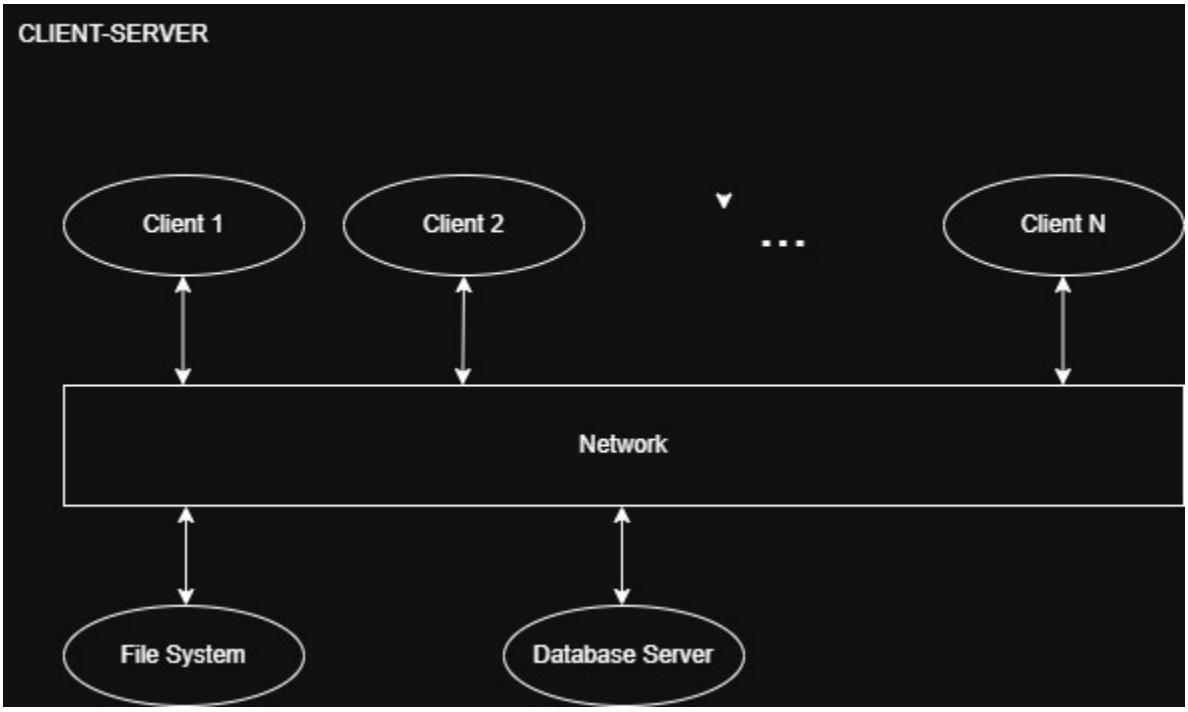
Нефункционални захтеви

Перформансе	Промене које начини један корисник треба да буду пропагиране до свих корисника за највише 2 секунде.
Сигурност	Логовање корисника. Сваки корисник има приступ само табли састанка ком је придружен.
Управљање ресурсима	Ослобађање меморије након брисања објеката са табле. Event processing не сме блокирати GUI. Мрежни проток оптимизован; шаљу се само делта промене по догађају.
Употребљивост	Интуитивни GUI • Канвас и алати за цртање морају бити лако препознатљиви • Дугмад јасно означена • Чет прозор једноставан и прегледан Једноставна интеракција • Повлачење линија, селектовање и брисање објеката интуитивно • Минималан број корака за основне операције Брза реакција система • Промене на табли и нове поруке се приказују брзо • Не блокира се GUI thread Приступачност • Текст читљив, контраст боја добар
Доступност	Теоретски систем може бити увек доступан (докле год сервер ради).
Поузданост	Губљење порука није дозвољено.
Скалабилност	Систем је предвиђен за 10-50 корисника.
Модификабилност	Мора да обезбеди једноставно додавање нових објеката.

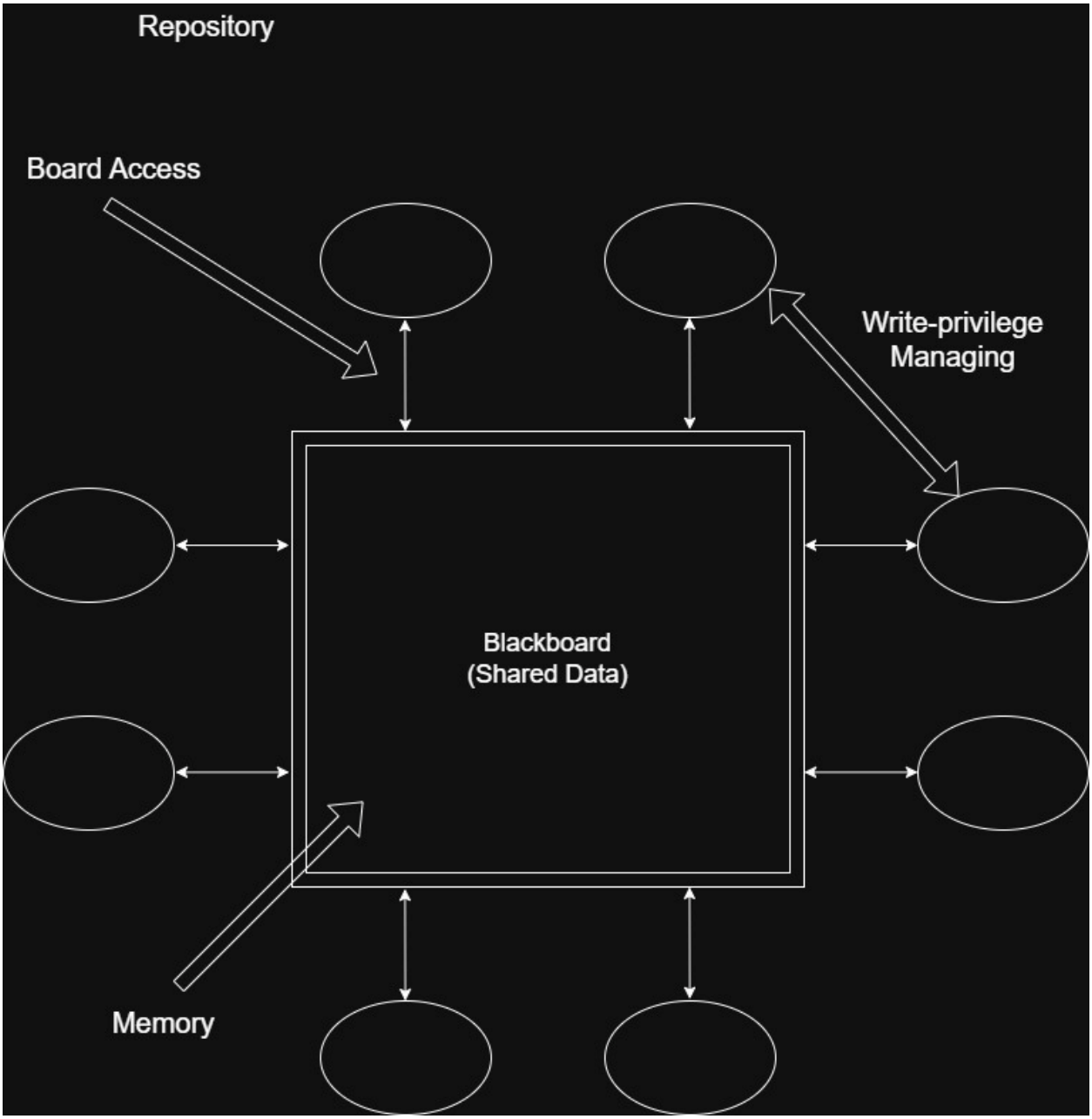
Архитектурни дизајн софтверског система

Архитектурни обрасци

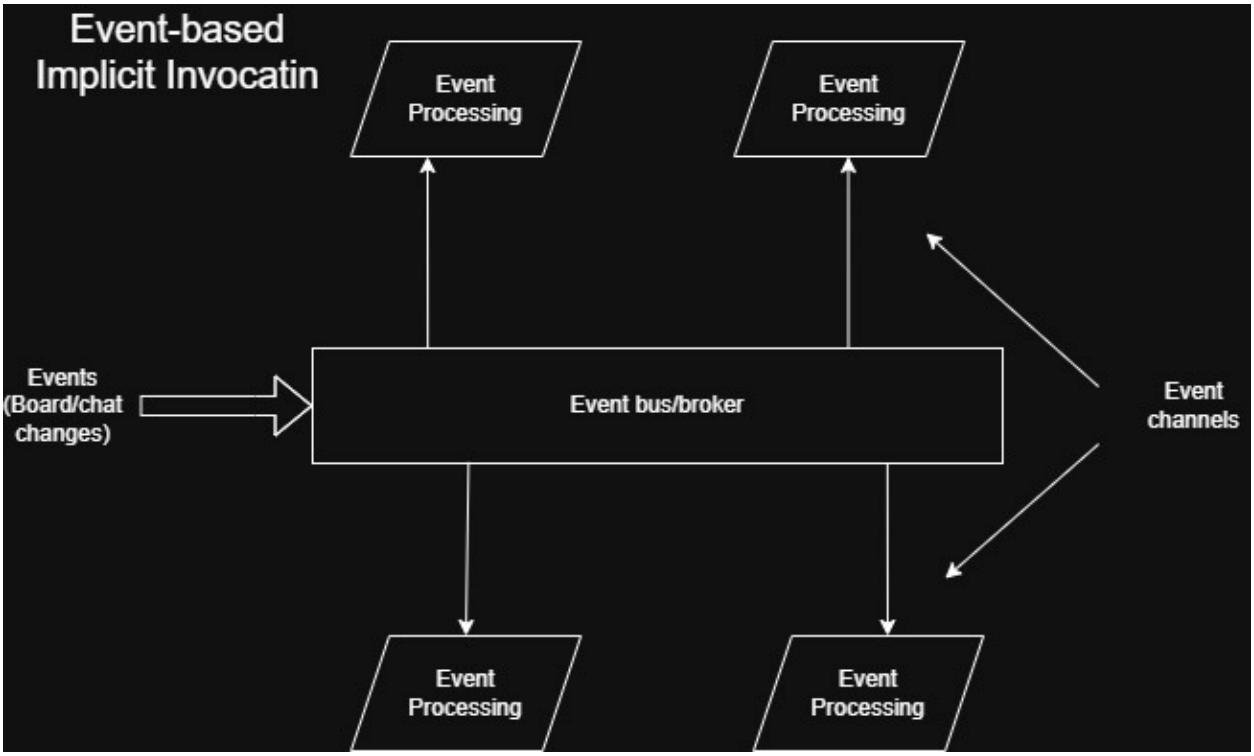
Client-Server – Наша апликација је дистрибуирана апликација.



Repository – Образац за складиштење (дељење приступа). Радимо **Blackboard** (активно складиште).



Event-based Implicit Invocation (push) – Кад клијент начини промену на табли имплицитно се обавештавају сви остали клијенти који приступају истој табли. Радимо **Event-based** подваријанту и то **push** типа јер је циљ да се сви обавесте кад се деси промена и да такорећи цртачка табла одлучује кад ће се то десити (зато је **push**).

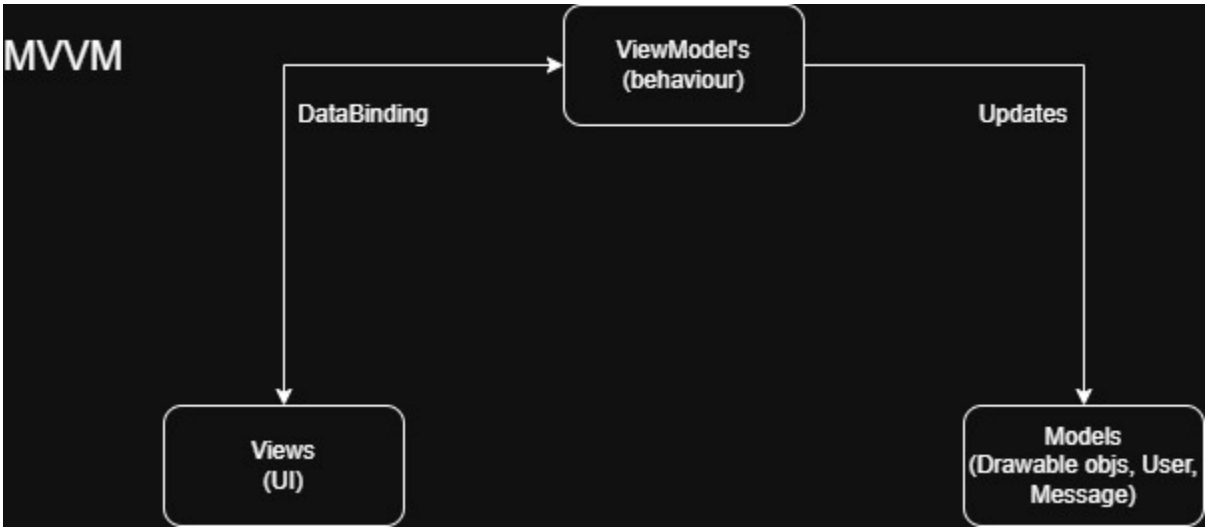


MVVC/MVM

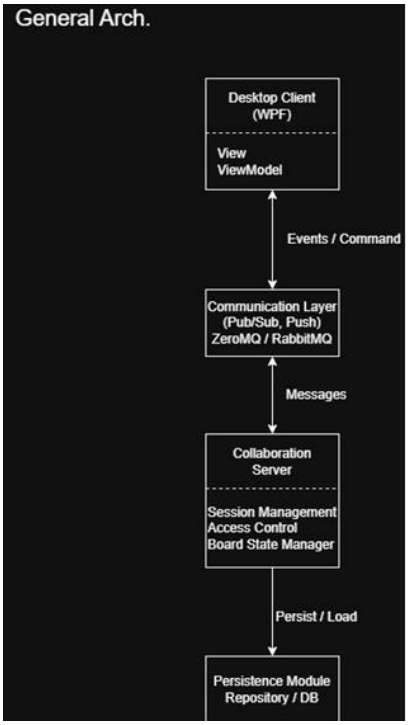
8. Корисник повуче линију на Canvas-у
   (акција мишем у View-у)
9. View региструје догађај и активира Command
   View не садржи пословну логику, већ позива DrawCommand дефинисан у ViewModel-у.
10. ViewModel прими Command
    ViewModel проверава да ли корисник има право цртања и припрема податке за нову линију.
11. ViewModel мења Model
    Креира се нови објекат типа Line, који се додаје у ObservableCollection<Shape>.
12. ViewModel шаље догађај серверу
    Промена се пакује у event поруку и шаље преко Communication Layer-а (publish/subscribe, push варијанта).

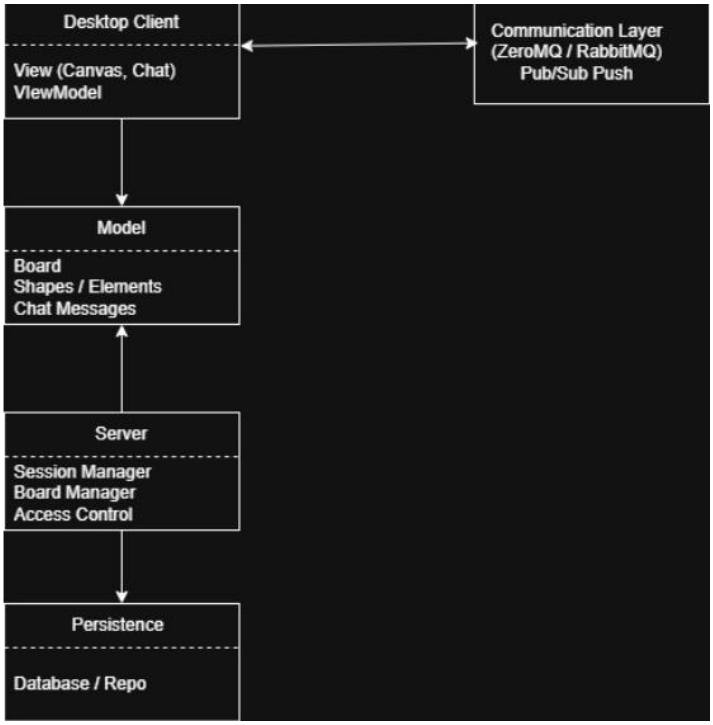
13. Model је промењен → View се аутоматски освежава
    Захваљујући data binding механизму, View аутоматски приказује нову линију без ручног освежавања.

14. Остали клијенти добијају догађај (push)
    Њихов ViewModel ажурира Model, након чега се њихов View аутоматски освежава.



Генерална архитектура система и структурни дијаграм





Техничка и пословна ограничења

Техничка ограничења

Клијентска апликација мора бити имплементирана као desktop апликација, коришћењем C# и WPF технологије.

За организацију корисничког интерфејса и логике мора се применити MVVM архитектурни образац.

Комуникација између клијената и сервера мора бити реализована коришћењем message-passing библиотеке или message broker-а (нпр. ZeroMQ или RabbitMQ).

Серверска компонента мора подржавати event-based publish/subscribe комуникацију у push варијанти.

Систем мора бити компатибилан са Windows оперативним системом.

Перзистенција података мора бити реализована коришћењем централизованог repository-ја / базе података.

GUI нит не сме бити блокирана мрежном комуникацијом или обрадом догађаја (асинхрони рад).

Пословна ограничења

Пословна ограничења дефинишу оквир у ком се систем развија и користи, независно од техничке имплементације.

Пословна ограничења система

Систем је намењен едукативној и демонстрационој употреби у оквиру академског пројекта.

Обим функционалности је ограничен на:

заједничко цртање UML/дијаграмских елемената

текстуални chat

управљање улогама корисника

Максималан број истовремених корисника у једној сесији је ограничен на 10–50 корисника.

Аудио и видео комуникација нису део основне функционалности система (могућа будућа надоградња).

Систем не захтева интеграцију са спољним комерцијалним сервисима.

Безбедносни механизми су основног нивоа (аутентикација и контроле приступа), без напредних enterprise механизама.

Рок за испоруку архитектурне фазе је дефинисан наставним планом и временским оквиром курса.

Изабрани апликациони оквири и библиотеке

1. Клијентска компонента (desktop апликација)

Језик: C#
Framework: WPF (Windows Presentation Foundation)

Разлог избора:

- Модеран графички кориснички интерфејс са подршком за Canvas, toolbar и стилизоване контроле.
- Једноставна имплементација **MVVM обрасца (Model–View–ViewModel)**, што омогућава јасну сепарацију GUI-ја и логике.
- Одлична подршка за **data binding**, што омогућава аутоматско освежавање View-а при променама у Model-у.

Обрасци који се примењују у клијенту:

- **MVVM образац:**
    - **Model:** чува стање табле, chat порука и корисничких права.
    - **ViewModel:** прима догађаје из View-а, ажурира Model и обрађује долазне event-е са сервера.
    - **View:** Canvas за цртање, chat прозор и toolbar, који се аутоматски освежавају при променама у Model-у.
- **Event-based push:**
  Клијент се претплаћује на догађаје са Blackboard сервера или message broker-а и реагује када стигну нови догађаји.

Библиотеке и алати:

- **Reactive Extensions (Rx.NET)** – за елегантно управљање догађајима и асинхроним токовима података.
- **Newtonsoft.Json** – за серијализацију објеката (цртежа и порука) приликом слања преко мреже.
- **ZeroMQ или RabbitMQ .NET клијент** – за комуникацију са сервером / broker-ом.

2. Серверска компонента (дистрибуирани сервер / Blackboard)

Језик: C#
Тип апликације: конзолна или сервисна (.NET)

Обрасци који се примењују:

- **Blackboard / Repository образац:**
  Централизовано активно складиште стања табле, историје цртежа и chat порука.
- **Implicit Invocation / Event push:**
  Сервер (Blackboard) одлучује када ће клијентима послати нове догађаје.

Библиотеке и алати:

- **ZeroMQ / RabbitMQ .NET клијент** – за дистрибуцију догађаја ка свим клијентима.
- **Entity Framework Core** – за перзистенцију података у локалној или удаљеној бази података.

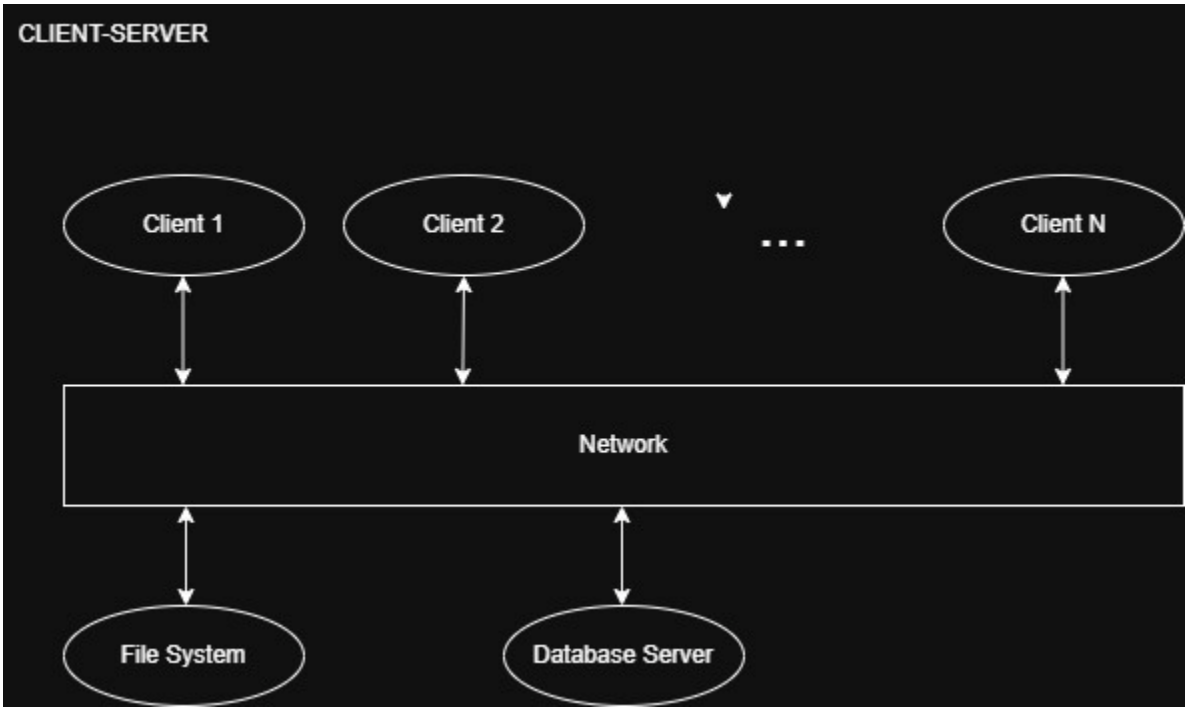
3. Додатне напомене

- **MVVM образац** се примењује искључиво на клијентску GUI логику, док сервер користи **Blackboard + Event push архитектуру**.
- Оваква комбинација омогућава:
  - Јасну модуларност и сепарацију одговорности
  - Једноставно проширење система (додавање нових типова објеката или догађаја)
  - Поуздану и real-time дистрибуцију догађаја свим корисницима

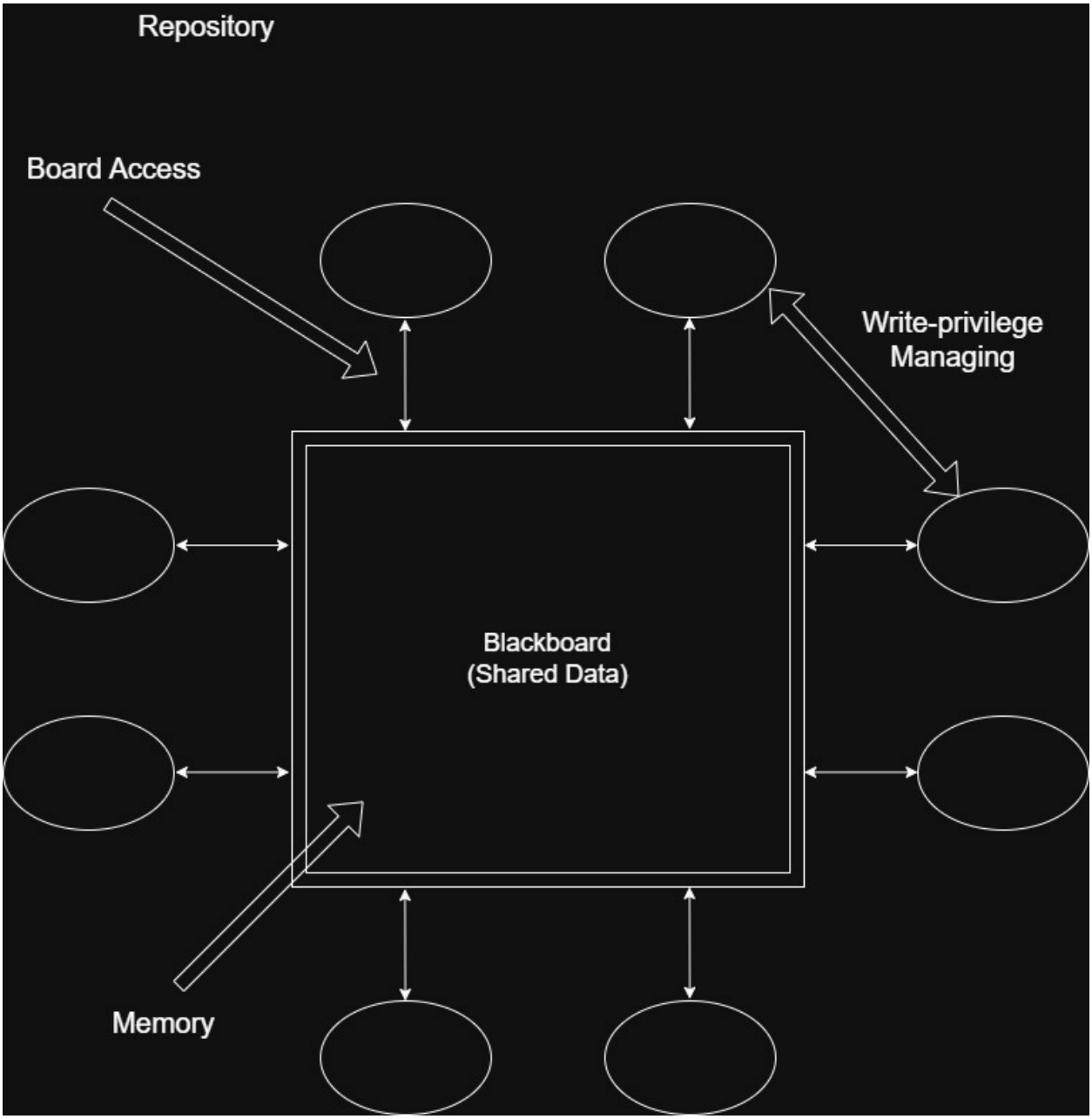
Архитектурни дизајн софтверског система

Архитектурни обрасци

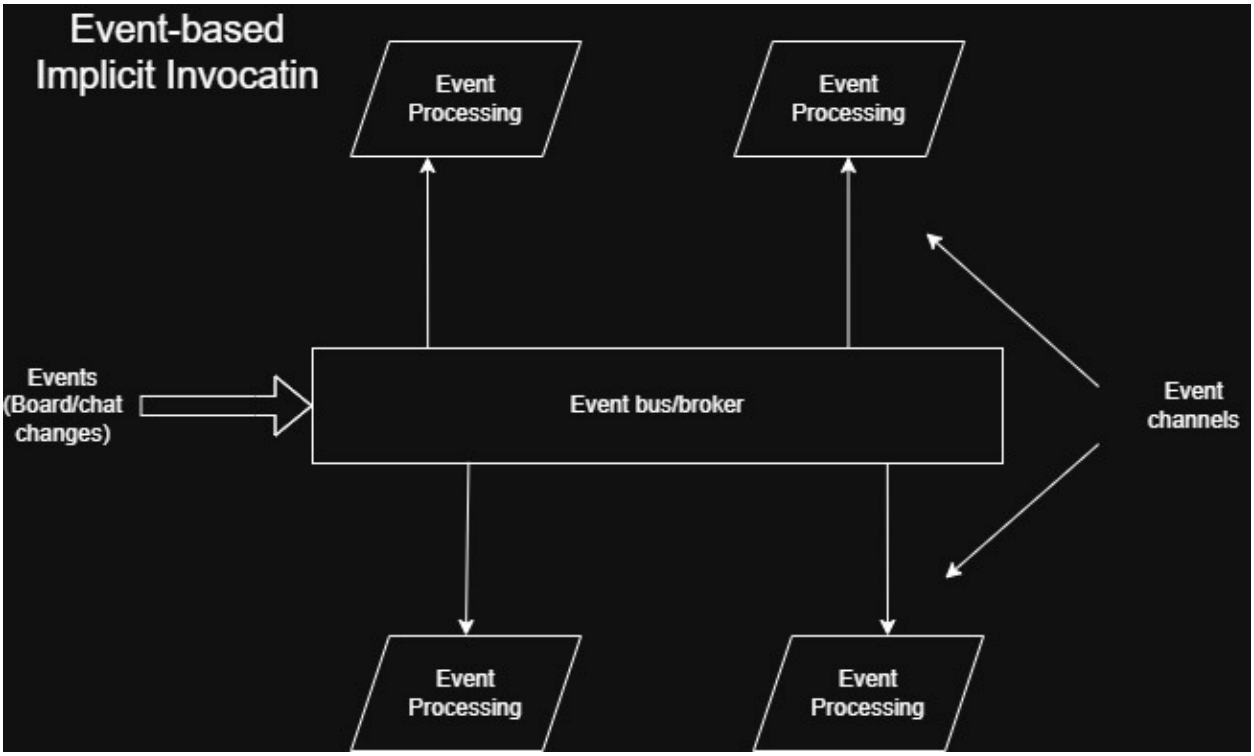
Client-Server – Наша апликација је дистрибуирана апликација.



Repository – Образац за складиштење (дељење приступа). Радимо **Blackboard** (активно складиште).



Event-based Implicit Invocation (push) – Кад клијент начини промену на табли имплицитно се обавештавају сви остали клијенти који приступају истој табли. Радимо **Event-based** подваријанту и то **push** типа јер је циљ да се сви обавесте кад се деси промена и да такорећи цртачка табла одлучује кад ће се то десити (зато је **push**).

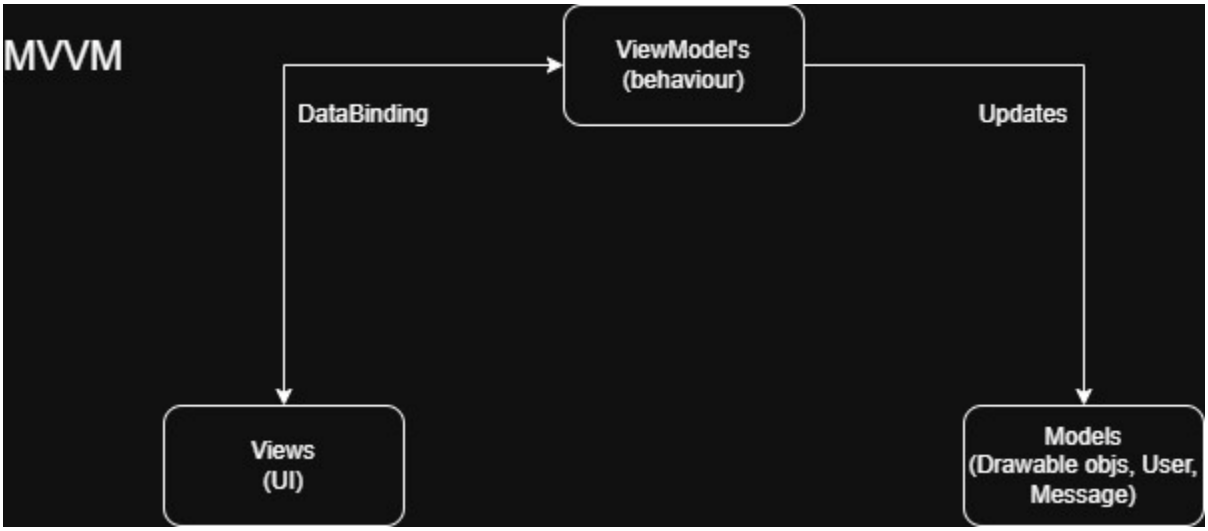


MVVC/MVM

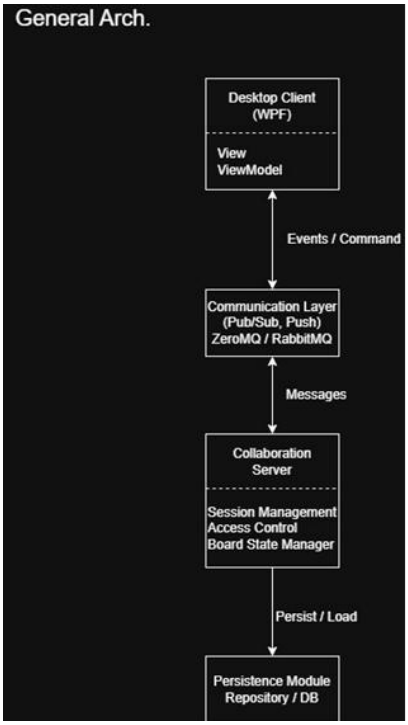
15. Корисник повуче линију на Canvas-у
    (акција мишем у View-у)
16. View региструје догађај и активира Command
    View не садржи пословну логику, већ позива DrawCommand дефинисан у ViewModel-у.
17. ViewModel прими Command
    ViewModel проверава да ли корисник има право цртања и припрема податке за нову линију.
18. ViewModel мења Model
    Креира се нови објекат типа Line, који се додаје у ObservableCollection<Shape>.
19. ViewModel шаље догађај серверу
    Промена се пакује у event поруку и шаље преко Communication Layer-а (publish/subscribe, push варијанта).

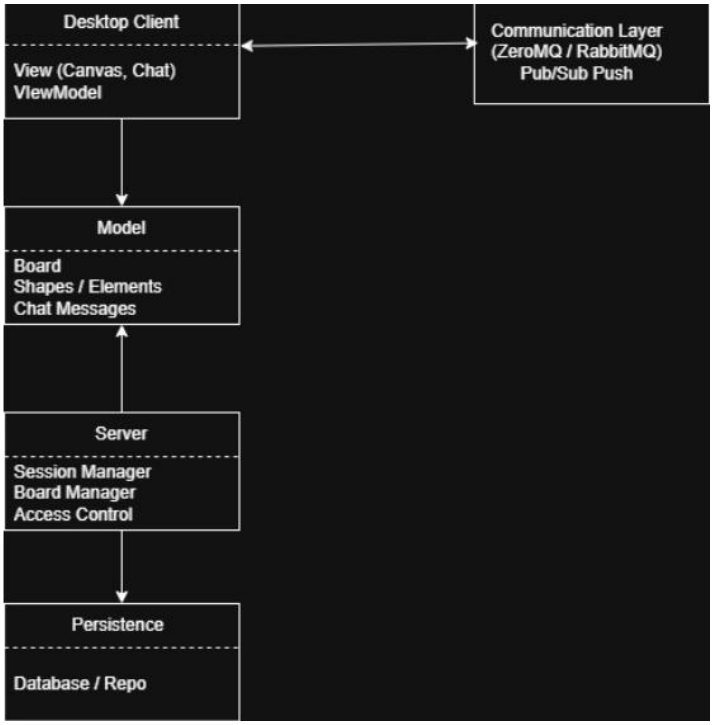
20. Model је промењен → View се аутоматски освежава
    Захваљујући data binding механизму, View аутоматски приказује нову линију без ручног освежавања.

21. Остали клијенти добијају догађај (push)
    Њихов ViewModel ажурира Model, након чега се њихов View аутоматски освежава.



Генерална архитектура система и структурни дијаграм





Техничка и пословна ограничења

Техничка ограничења

Клијентска апликација мора бити имплементирана као desktop апликација, коришћењем C# и WPF технологије.

За организацију корисничког интерфејса и логике мора се применити MVVM архитектурни образац.

Комуникација између клијената и сервера мора бити реализована коришћењем message-passing библиотеке или message broker-а (нпр. ZeroMQ или RabbitMQ).

Серверска компонента мора подржавати event-based publish/subscribe комуникацију у push варијанти.

Систем мора бити компатибилан са Windows оперативним системом.

Перзистенција података мора бити реализована коришћењем централизованог repository-ја / базе података.

GUI нит не сме бити блокирана мрежном комуникацијом или обрадом догађаја (асинхрони рад).

Пословна ограничења

Пословна ограничења дефинишу оквир у ком се систем развија и користи, независно од техничке имплементације.

Пословна ограничења система

Систем је намењен едукативној и демонстрационој употреби у оквиру академског пројекта.

Обим функционалности је ограничен на:

заједничко цртање UML/дијаграмских елемената

текстуални chat

управљање улогама корисника

Максималан број истовремених корисника у једној сесији је ограничен на 10–50 корисника.

Аудио и видео комуникација нису део основне функционалности система (могућа будућа надоградња).

Систем не захтева интеграцију са спољним комерцијалним сервисима.

Безбедносни механизми су основног нивоа (аутентикација и контроле приступа), без напредних enterprise механизама.

Рок за испоруку архитектурне фазе је дефинисан наставним планом и временским оквиром курса.

Изабрани апликациони оквири и библиотеке

1. Клијентска компонента (desktop апликација)

Језик: C#
Framework: WPF (Windows Presentation Foundation)

Разлог избора:

- Модеран графички кориснички интерфејс са подршком за Canvas, toolbar и стилизоване контроле.
- Једноставна имплементација **MVVM обрасца (Model–View–ViewModel)**, што омогућава јасну сепарацију GUI-ја и логике.
- Одлична подршка за **data binding**, што омогућава аутоматско освежавање View-а при променама у Model-у.

Обрасци који се примењују у клијенту:

- **MVVM образац:**
    - **Model:** чува стање табле, chat порука и корисничких права.
    - **ViewModel:** прима догађаје из View-а, ажурира Model и обрађује долазне event-е са сервера.
    - **View:** Canvas за цртање, chat прозор и toolbar, који се аутоматски освежавају при променама у Model-у.
- **Event-based push:**
  Клијент се претплаћује на догађаје са Blackboard сервера или message broker-а и реагује када стигну нови догађаји.

Библиотеке и алати:

- **Reactive Extensions (Rx.NET)** – за елегантно управљање догађајима и асинхроним токовима података.
- **Newtonsoft.Json** – за серијализацију објеката (цртежа и порука) приликом слања преко мреже.
- **ZeroMQ или RabbitMQ .NET клијент** – за комуникацију са сервером / broker-ом.

2. Серверска компонента (дистрибуирани сервер / Blackboard)

Језик: C#
Тип апликације: конзолна или сервисна (.NET)

Обрасци који се примењују:

- **Blackboard / Repository образац:**
  Централизовано активно складиште стања табле, историје цртежа и chat порука.
- **Implicit Invocation / Event push:**
  Сервер (Blackboard) одлучује када ће клијентима послати нове догађаје.

Библиотеке и алати:

- **ZeroMQ / RabbitMQ .NET клијент** – за дистрибуцију догађаја ка свим клијентима.
- **Entity Framework Core** – за перзистенцију података у локалној или удаљеној бази података.

3. Додатне напомене

- **MVVM образац** се примењује искључиво на клијентску GUI логику, док сервер користи **Blackboard + Event push архитектуру**.
- Оваква комбинација омогућава:
  - Јасну модуларност и сепарацију одговорности
  - Једноставно проширење система (додавање нових типова објеката или догађаја)
  - Поуздану и real-time дистрибуцију догађаја свим корисницима